
PROYECTO FINAL DE CARRERA



UNIVERSITAT
ROVIRA I VIRGILI



Sistema MultiAgente para la extracción de información desde el World Wide Web

Marc Sànchez Artigas, msa.si@estudiants.urv.es
Ingeniería Técnica en Informática de Sistemas

Directores del proyecto:
Arantza Aldea
David Riaño

Escola Tècnica Superior d'Enginyeria (ETSE)
Universitat Rovira i Virgili (URV)
<http://www.etse.urv.es>

Curso 2001-2002

Índice

1	Introducción	3
1.1	Objetivos	4
1.2	Motivación	4
1.3	Descripción del proyecto	5
1.3.1	Ontologías	6
1.3.2	Agente de Internet($A_{internet}$)	7
1.3.3	Agente Coordinador(A_{coord})	7
1.3.4	Intercambio de datos	9
2	Ontologías Web	10
2.1	XML	12
2.2	RDF	16
2.2.1	Qué es RDF?	16
2.2.2	Descripción de RDF	17
2.2.3	Vocabularios y Esquemas RDF	19
2.2.4	Características de RDF	22
3	Agentes y Sistemas MultiAgentes	23
3.1	Definición de Agente	23
3.1.1	Atributos	24
3.1.2	Tipos de Agentes.	26
3.2	Arquitectura de SMA	28
3.2.1	Directory Facilitator(DF)	30
3.2.2	Agent Management System(AMS)	31
3.2.3	Agent Communication Channel(ACC)	31
3.2.4	Agent Platform	32
3.3	Lenguaje de Comunicación	34
3.3.1	Protocolos de Comunicación	36
4	JADE: entorno de desarrollo de SMA	40
4.1	Implementación de un agente	42
4.2	Comportamientos	44
4.3	Protocolos de interacción	48

5 Justificación de las técnicas utilizadas	50
5.1 Porqué RDF?	50
5.2 Porqué RDF y no XML?	51
5.3 Porqué Agentes?	55
5.4 Conclusiones	56
6 Implementación del SMA	57
6.1 JENA API	57
6.1.1 Sintaxis RDQL	58
6.2 Agente de Internet($A_{internet}$)	59
6.2.1 El proceso P_{get}	63
6.2.2 El proceso $P_{analysis}$	64
6.2.3 Response Ontology	69
6.3 Agente Coordinador(A_{coord})	70
6.3.1 El proceso $P_{splitter}$	71
6.3.2 El proceso $P_{controller}$	74
6.3.3 El proceso $P_{assembler}$	75
7 Resultados	76
8 Conclusiones y Trabajo Futuro	98
10 Referencias	99

1. Introducción

Nuestro mundo ha cambiado mucho en las últimas décadas. Se ha pasado de disponer de poca información, prácticamente la transmitida por los medios de comunicación tradicionales: prensa, televisión, radio etc. , a una sociedad, denominada de la *información*, donde los individuos constituyentes, somos incapaces de poder procesar todo el volumen de información disponible. Especialmente, el mundo del **World Wide Web**, ha permitido, en la última década, disponer de un volumen de información, que hasta el momento era impensable por vías tradicionales. Sin embargo, y en gran medida a su rápido crecimiento, también, se ha convertido en uno de los medios más complicado de manipular a todas las escalas. Es difícil de acceder a la información disponible.

Para entender el problema, pongamos el siguiente ejemplo. Supongamos que estamos buscando información sobre el proceso de **destilación**. Para encontrar dicha información recorreremos a un motor de búsqueda tal como *Google*. Introduciremos la cadena de texto **“destilación”** y esperaremos que nos devuelva una multitud de enlaces a páginas. Muchas de ellas, estarán apenas relacionadas con el proceso de destilación, y varias operaciones de *limpieza*, de forma manual, se habrán de hacer para filtrar la información útil. Además, si quisiéramos buscar información extra relacionada con la *destilación*, como la teoría, los elementos reactivos, el tipo de columnas, equipo especializado etc. nos encontraríamos que en los motores actuales, habríamos de buscar los términos de forma individual; puesto que la reunión de términos no resultaría factible (pocas páginas contienen todos los términos). Todo este proceso de búsqueda podría llevar a un usuario a horas de trabajo.

Entonces, la solución correría a cargo de diseñar aplicaciones automáticas, que ayudasen a los expertos a descubrir conocimiento sin necesidad de una intervención humana continuada. De esta forma, el objetivo de este proyecto, es desarrollar un buscador, que mediante una representación con términos y sus relaciones, pueda extraer conocimiento de la web(el cual no se podría extraer usando los términos de forma individual) de manera automática.

El dominio de conocimiento habría de ser independiente de la implementación. El sistema simplemente recibiría una terminología de entrada construida por expertos y esperaría recibir resultados. Además, para permitir hacer la aplicación distribuida y dotarla de autonomía, usaríamos la capacidad de los sistemas Multiagente. Más adelante, explicaremos que son y que relación guardarán con nuestro buscador.

Una vez explicada esta introducción, pasaremos a describir cuáles son los objetivos de este proyecto. Después, haremos una descripción de qué se quiere implementar.

1.1 Objetivos

Los principales objetivos del proyecto son:

- ❑ Descubrir conocimiento en un dominio particular. Por ejemplo, recobrar toda la información disponible sobre **destilación**(dominio de Química).
- ❑ Hacer una implementación independiente del dominio; los datos (buscados y encontrados) deben ser representados en una forma estándar.
- ❑ El usuario pueda configurar la aplicación para satisfacer su necesidades. Esto es, que el usuario pueda decidir a que tipo de información dar relevancia según sus criterios. Por ejemplo, podría interesar buscar todas aquellas páginas cuyo título contuviera la palabra *destilación*.

1.2 Motivación

El desarrollo de este proyecto ha venido motivado por ser una parte constitutiva del proyecto europeo h-techsight. H-techsight (IST 2001-33174) está financiado por la Unión Europea y comenzó en Mayo del 2002 con una duración de dos años. En él colaboran los departamentos de Ingeniería Química e Ingeniería Informática de nuestra universidad. El objetivo de este proyecto es desarrollar una herramienta que ayude a pequeñas y medianas empresas a acceder a la información publicada en la red. La herramienta se ocupará de ayudar a organizar la búsqueda, encontrando la información de una manera inteligente, eliminando todo tipo de links innecesarios, ordenando la información, procesándola y algo muy importante para las empresas, percibiendo dinámicamente nuevos conceptos o links que el usuario desconocía.

En este proyecto se pueden distinguir dos etapas. En la primera etapa se desarrollará una herramienta que permita al usuario representar el conocimiento que posee sobre el tema del que se quiere buscar información y se creará un buscador inteligente que encuentre información y la organice para presentarla al usuario. En la segunda etapa se deberá procesar la información obtenida para encontrar nuevos conceptos que son ignorados por el usuario.

Por tanto, este proyecto se ha desarrollado para proporcionar un prototipo de buscador inteligente en el cual basar la primera parte de este proyecto.

1.3 Descripción del Proyecto

Como hemos mencionado hace dos secciones, la implementación del buscador se hará usando un *Sistema MultiAgente o SMA*. Si el lector desconoce los detalles o características de estos sistemas, le invitamos a leer el capítulo 3 de esta documentación. Así pues, esta sección pretende explicar los agentes[**sección 3.1**] involucrados en el sistema, sus tareas y los protocolos de comunicación usados por ellos. El primer paso es especificar que agentes formarán parte del sistema. Después, detallaremos que protocolos de comunicación y estructuras de datos se usarán.

Los agentes a implementar son:

- ❑ Agente de Internet($A_{internet}$)
- ❑ Agente Coordinador(A_{coord})

Las relaciones entre ellos se muestra en la Fig.1. En ella podemos observar las relaciones entre los diferentes agentes del sistema y los tipos de comunicación disponibles. Hay dos tipos de comunicación: la comunicación interagente y la comunicación no interagente, esto es, la comunicación con un usuario fuera del sistema. Los mensajes intercambiables entre los agentes se denominan ACL [**Agent Communication Message: sección 3.3**]. Los mensajes usados en la comunicación no interagente se basan en llamadas a métodos realizadas a partir de una interfaz gráfica. El protocolo de comunicación elegido para la comunicación entre agentes es el protocolo *FIPA-Request*[**sección 3.3.1**], como muestra el diagrama.

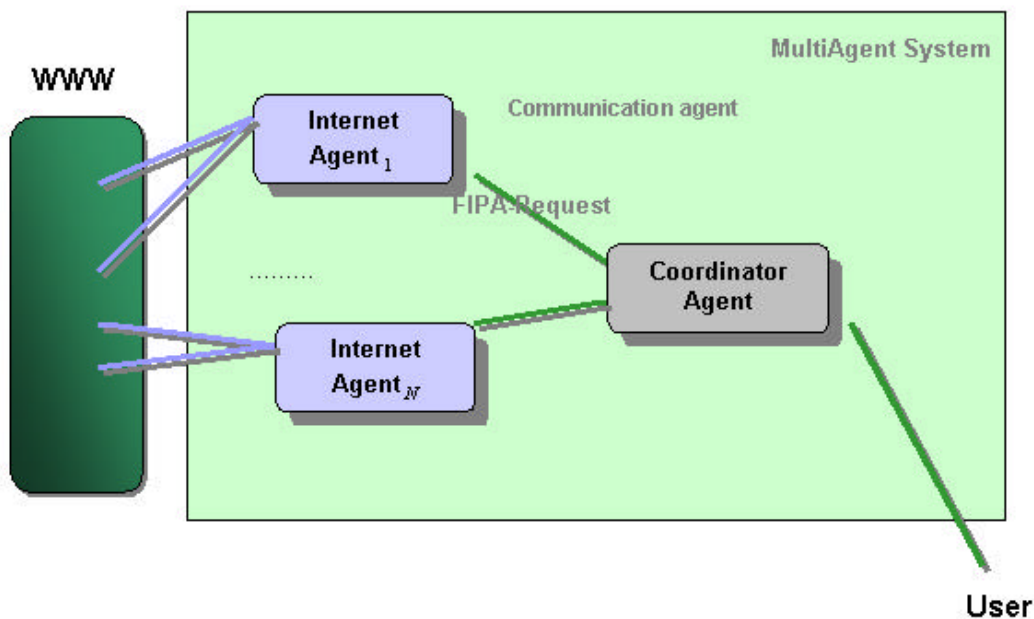


Figura 1. Sistema MultiAgente para la extracción de información del WWW.

1.3.1 Ontologías

Antes de empezar a discutir como trabajan los agentes involucrados en el sistema, sería necesario introducir cuáles son las ontologías que serán usadas por ellos. Entonces, la primera pregunta que realizará todo lector es ¿Qué es una ontología?. Más adelante, ya en el siguiente capítulo, discutiremos en profundidad este asunto. De momento, diremos que una ontología es forma de representación de conocimiento. Así pues, las ontologías que se intercambiarán los agentes en nuestro sistema, servirán para expresar el “conocimiento” que se manejará. Esto es, el conocimiento referente tanto a los términos de búsqueda como a sus resultados. Evidentemente, las ontologías y sus implementaciones no serán siempre las mismas. Habrá cuatro tipos de ontologías: **query ontology**, **response ontology**, **domain ontology**, **information ontology**.

- ❑ **query ontology:** es la entrada de los $A_{internet}$. Es enviada por el A_{coord} .
- ❑ **response ontology:** es la salida de los $A_{internet}$.
- ❑ **domain ontology:** Es la ontología creado por el experto. Está expresada como una jerarquía de clases. Cada clase o categoría está compuesta de términos denominados propiedades. Las relaciones entre las distintas clases están etiquetadas.
- ❑ **information ontology:** Es la unión de todas las ontologías de respuesta creada por el A_{coord} .

La implementación de las ontologías se hará usando RDF[sección 2.2]. La sintaxis de RDF es similar a XML[sección 2.1], pero en realidad es una extensión de

ésta. La última ontología se representará, además, de forma visual en formato HTML en la interfaz gráfica del A_{coord} .

1.3.2 Agente de Internet($A_{internet}$)

La principal función de este agente es recobrar información de la web y elaborar una representación de la información encontrada como una colección ordenada de páginas visitadas. Este agente recibe como parámetro el conjunto de términos que formarán la materia prima del *motor de búsqueda*.

Este agente empieza la búsqueda a partir de una serie de sitios que ha conocido previamente, por ejemplo, a partir de un portal como **Google**. El proceso que lleva a cabo esta tarea se denominará P_{get} .

Una vez dispone de enlaces a los primeros sitios web y de los términos de búsqueda [**query ontology**], empieza a realizar la búsqueda. El agente conoce el **deadline** y controla el tiempo de entrega de resultados. Una vez llega el **deadline**, el agente envía una señal para que finalice el proceso de análisis. Entonces, el agente empieza a elaborar la respuesta final juntando todas las instancias de las páginas visitadas.

Además, este agente dispone de dos conjuntos de datos usados para manipular la páginas visitadas y los sitios web para visitar. De este modo, conseguimos eliminar redundancias entre los nuevos sitios por visitar y los visitados en el pasado.

El proceso de análisis o $P_{analysis}$ extrae toda la información disponible en un Website, la analiza según la ontología de entrada y genera una ontología de respuesta.

La **query ontology** es una colección de elementos básicos [**clases**] y sus propiedades. Cada elemento de esta ontología será analizado en cada web. El resultado de este análisis será una colección de instancias de esta ontología.

1.3.3 Agente Coordinador(A_{coord})

El principal objetivo de este agente es componer una representación del conocimiento obtenido por todos los Agentes de Internet.

El agente será configurado, vía interfaz gráfica, para empezar a trabajar. La configuración comprenderá: el tiempo de búsqueda o **deadline**, la ontología de entrada con los términos y sus relaciones, y el **threshold** o umbral a partir del cual los Agentes de Internet filtrarán las páginas visitadas. Esta configuración puede ser modificada posteriormente, y en tiempo de ejecución, añadiendo y/o eliminando nuevas propiedades sobre los términos de entrada. El resto de parámetros no podrán ser modificados en tiempo de ejecución.

En el sistema hay varios $A_{internet}$ que están esperando peticiones de búsqueda. Este agente se ocupará de repartir trabajo entre los diferentes agentes, los cuales estarán ociosos hasta ese momento. (ver Figura 2.)

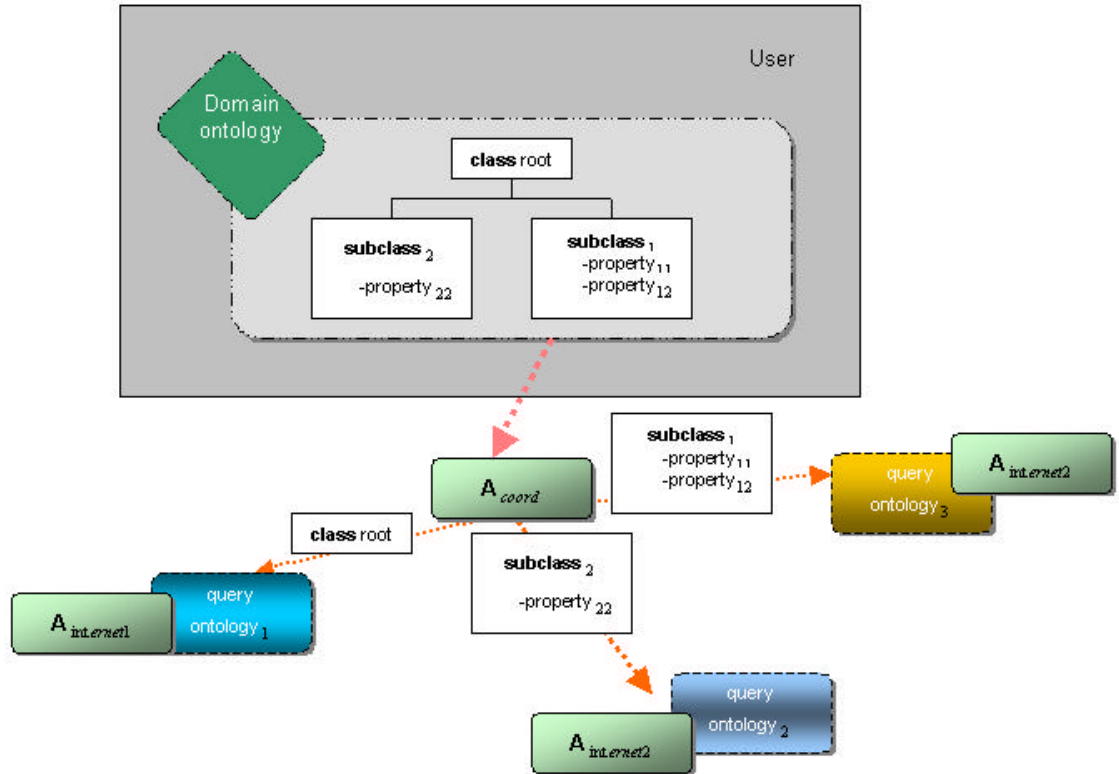


Figura 2. A_{coord} separa la **domain ontology** en varias piezas las cuales son distribuidas a los $A_{internet}$.

Este agente recibe: la ontología compuesta por los expertos sobre un dominio o una área de interés y las ontologías de respuesta de los diferentes $A_{internet}$. Con estos datos ha de ser capaz de generar una base de conocimiento, que se visualizará en algún formato estándar.

El agente recibirá el **deadline** de entrada, se lo comunicará a los diferentes Agentes de Internet y esperará recibir una vez excedido este período de tiempo, las diferentes ontologías de respuesta. El **umbral** también será transmitido a los $A_{internet}$, para que hagan el filtrado de las páginas visitadas justo después de ser analizadas.

La figura 3. muestra un resumen de los procesos que realiza el A_{coord} . En este diagrama se observa como este agente consta de tres procesos:

- $P_{controller}$: responsable de asignar las tareas a los agentes de Internet.

- $P_{assembler}$: proceso que se ocupa de componer las respuestas de todos los agentes y representar la base de conocimiento, en un formato estándar.
- $P_{splitter}$: proceso que se ocupa de dividir los términos de ontología de entrada, en porciones individuales, que los $A_{internet}$ procesarán.

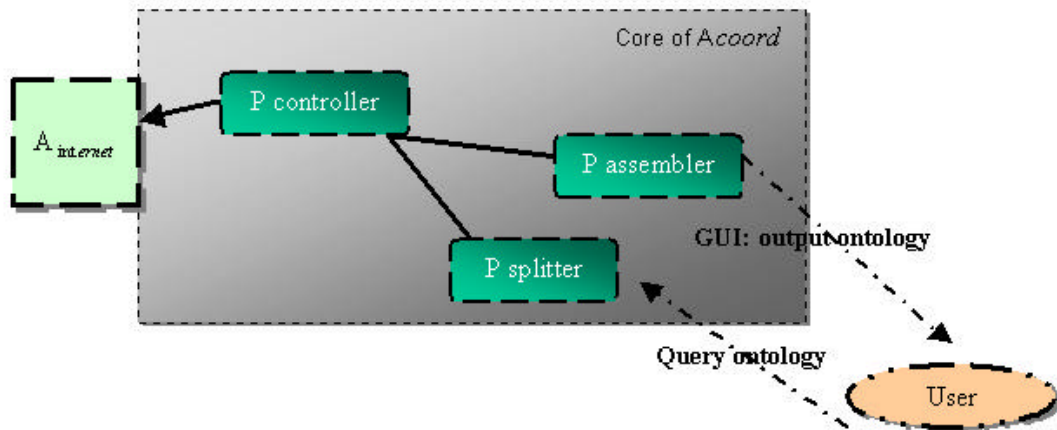


Figura 3. Los procesos involucrados en A_{coord} .

1.3.4 Intercambio de Datos

Los datos que se intercambiarán los agentes, serán los datos que provienen del A_{coord} y se dirigen a los Agentes de Internet.

Dirección	Datos
A_{coord} hacia $A_{internet}$	□ La lista de términos de búsqueda. □ Tiempo de trabajo [<i>deadline</i>]. □ Umbral de filtro [<i>threshold</i>].

2. ONTOLOGÍAS WEB

Este capítulo está destinado hacer una breve introducción sobre qué son las ontologías. Este capítulo se ha incluido en el proyecto por dos motivos:

- ❑ **Justificación de su uso:** para que nuestros lectores puedan entender en los siguientes capítulos porque se han utilizado ontologías.
- ❑ **Comprensión:** para que los lectores puedan comprender, con mayor facilidad, como funcionan los lenguajes que las representan.

De esta forma, cualquier lector de este proyecto estará capacitado para entender cuál es la funcionalidad de cada una de las ontologías del sistema(**query ontology**, **information ontology** etc.).

El tratamiento de la temática de este capítulo se ha dividido en dos partes. En la primera parte, se expone de forma general qué es una ontología. En la segunda, se comentan los lenguajes de representación involucrados en el sistema. Esta última parte se corresponde con las **secciones 2.1 y 2.2**.

Una vez comentado como se distribuirá el contenido de este capítulo, vayamos a empezar:

Llegados a este punto todo lector se debería de preguntar *¿Qué es una ontología?*. La respuesta sería la siguiente:

<< Una **ontología** es conjunto de términos compartidos por una comunidad para intercambiar datos dentro de un dominio de aplicación >>.

Sin embargo, para la mayoría de los lectores esta definición resultará bastante inútil. Así pues, para definir que es una ontología, quizás sería más interesante intentar pensar para que se usan comúnmente, y que relacionan guardan con el **World Wide Web**. Para dicha tarea, propondremos un ejemplo sencillo:

Supongamos que dos entidades bancarias quieren mandarse información sobre las cuentas de sus clientes a través de la web. Entonces, el primer paso que toman los directores es estudiar que se puede necesitar para comunicar ambos bancos. Finalmente, y tras largas reuniones, los directores deciden que se necesitan dos elementos. En primer lugar, ordenadores (que los clientes pagarán de sus hipotecas), y en segundo lugar, que ambos bancos definan y usen una terminología común, que permita a sus nuevas computadoras entender los mensajes recibidos y generar mensajes *con sentido*. Es decir, los directores comprenden que para comunicar los bancos no basta con mandar bits, sino que es necesario que la información transmitida sea *interpretable* por sus máquinas. Esto implica que los mensajes deban utilizar términos que den significado al contenido, como por ejemplo, *titular*, *saldo*, *número de cuenta* etc. Sin estos términos, el contenido no sería entendible por las computadoras ni incluso por cualquier empleado del banco. Si dieran a un empleado una lista con una secuencia de números sin más información, no podríamos esperar que este empleado fuera capaz de saber si los números son teléfonos o cuentas bancarias. En este contexto, una terminología hubiera

ayudado a determinar de manera inequívoca el contenido de la lista. Entonces, los directores se ven con la necesidad de declarar una terminología que de significado a sus mensajes. Finalmente, cuando los directores definan su terminología, sin darse cuenta, habrán definido una *ontología* cuyo dominio de aplicación es el dominio *bancario*.

La especificación de cualquier ontología necesita de algún lenguaje de representación. Estos lenguajes sirven para definir términos y señalar las relaciones que guardan entre ellos. De lenguajes de representación hay muchos. Algunos como **KIF-based Ontololingua**, **Loom**, **Frame-Logic** basan su expresividad en la lógica de primer orden. Sin embargo, para aplicaciones en la web es importante poseer un lenguaje con sintaxis estandarizada. Debido a que **XML** [**Extensible Markup Language**] [1] se ha convertido en un lenguaje estándar para el *intercambio de datos en la web*, se ha hecho común el intercambio de ontologías expresadas en sintaxis **XML**. Este requisito ha llevado a usar lenguajes basados en sintaxis **XML**. Algunos de ellos son **RDF** [**Resource Description FrameWork**], **OIL** [**Ontology Interchange Language**] y **DAML+OIL**. **RDF** tiene grandes garantías de convertirse en un estándar aceptado. **DAML + OIL**, desarrollado por un grupo de científicos europeos y americanos, se ha convertido en el sucesor de **OIL**. Sin embargo, antes de empezar hablar de todos estos lenguajes de representación, sería necesario dar una breve pincelada sobre que es **XML**. En la siguiente sección hablaremos con más detalle de este lenguaje.

XML [**Extensible Markup Language**] es un lenguaje marcado, basado en una sintaxis similar a **HTML**[†], que permite a los usuarios definir sus propias etiquetas y estructuras de documentos. El potencial que ofrece **XML** es utilizado para el *intercambio de datos*, la creación de protocolos e infraestructuras para registros de empresas(**ebXML** y **UDDI**), y la adaptación de interfaces de presentación a múltiples dispositivos (**WAP**, **PDA** etc.).

Actualmente, organizaciones y empresas cooperan en la definición de conjuntos de etiquetas para sectores de la industria específicos (*turismo, banca, transporte, medicina*,...), coordinados por el organismo sin ánimo de lucro **OASIS**.

A pesar de todo, **XML**, adolece de ciertas deficiencias. **XML** permite a los usuarios definir etiquetas como por ejemplo, **<AUTHOR>**, las cuales parecen estar dotadas de cierto significado. Sin embargo, des del punto de vista computacional etiquetas como **<AUTHOR>** tienen el mismo significado que una etiqueta como **<H1>**. Un computador simplemente no conoce, que es un autor, ni si el concepto está relacionado con una persona por ejemplo. *XML puede ayudar a los humanos a predecir que información puede estar contenida entre etiquetas como en el caso de **<AUTHOR></AUTHOR>**, pero sólo puede ayudar.* Para un procesador **XML**, **<AUTHOR>** y **<BOOK>** son totalmente iguales, no tienen ningún significado. Esto,

[†] **HTML** [**Hypertext Markup Language**] es el lenguaje de marcado que se utiliza para codificar el formato de presentación y enlaces de hipertexto de las páginas **Web**. Se basa en un sistema de etiquetas estandarizadas (tipo **<H1>** y **<BODY>**), el significado y la interpretación de las cuales son establecidas de forma universal por el **Consorcio World Wide Web**[2].

como veremos en secciones posteriores[*capítulo 5*], será determinante para decidir porque se desestimó para este proyecto.

2.1 XML

En la sección anterior, se ha discutido brevemente sobre que es *XML*. Sin embargo, para entender en profundidad que significa, deberíamos empezar viendo más bien en qué consiste y cuál es su relación con el lenguaje *HTML* y *SGML*[‡].

XML se diseñó como consecuencia del gran crecimiento de Internet, los intereses comerciales y la necesidad de poder realizar páginas Web vistosas. En aquel momento, *HTML* se había convertido en el mecanismo de presentación de documentos más extendido de la historia. Todas las grandes firmas se anunciaban en Internet a través de páginas *HTML* y su número no paraba de crecer y crecer. Sin embargo, todo este éxito desembocó en una paradoja. Produjo que evolucionará de forma muy rápida y no siempre tomando el camino más adecuado. De esta forma, *HTML* mantuvo su rigidez inicial y se convirtió en un lenguaje limitado. En un lenguaje de presentación de información directamente para humanos, pero difícil de procesar por programas informáticos. *HTML* permitía indicar si un tipo de letra era azul o si era de un tamaño determinado, pero era incapaz de decir si lo que estaba mostrando era el título de un libro o el precio de un artículo. Por este motivo nació *XML*. *XML* permitió describir aquello que etiquetaba.

En la actualidad, *HTML* y *XML* conviven en armonía. *XML* no se ha convertido para nada en el sustituto de *HTML* ni en ninguna extensión.

Para entender de forma gráfica el porqué del nacimiento de *XML*, proponemos el siguiente ejemplo esclarecedor:

Mostramos como Amazon presenta en su web información sobre libros. En *HTML* el código es el siguiente:

```
<p>
<dt>
<b>
<a href="/exec/obidos/ASIN/0764531999/qid=919015337">
Xml : Extensible Markup Language</a></b> ~
<NOBR><font color=#990033>Usually ships in 24 hours</font></NOBR>
<dd> Elliott Rusty Harold / Paperback / Published 1998
<br>
Our Price: $31.99 ~
<NOBR><font color=#990033>You Save: $8.00 (20%)</font></NOBR>
<br>
```

[‡] *SGML* [*Standard Generalized Markup Language, ISO 8879*] es el estándar internacional para la definición de la estructura y el contenido de diferentes tipos de documentos electrónicos.

```
<a href="/exec/obidos/ASIN/0764531999/qid=919015337">
<i>Read more about this title...</i></a>
```

En XML se podría escribir de la siguiente forma:

```
<?xml version="1.0"?>
<libro>
  <titulo>Xml: Extensible Markup Language</titulo>
  <disponible tiempo="24" unidad="hours" />
  <autor>Elliott Rusty Harold</autor>
  <formato>Paperback</formato>
  <publicacion>1998</publicacion>
  <precio cantidad="31.99" moneda="dolar" />
  <descuento cantidad="20" />
  <enlacelibro href="/exec/obidos/ASIN/0764531999/qid=919015337" />
</libro>
```

Evidentemente, no es necesario ser un programador informático para saber que trabajar sobre el segundo ejemplo es mucho más sencillo que en el primero.

En definitiva, el uso de XML permitió disponer de un acceso más rápido y eficiente a la información, facilitando el intercambio de información y la cooperación entre la empresas.

Una vez, mostrado que supuso la aparición de *XML*, deberíamos empezar a exponer algunas propiedades de *XML* y concretar más exactamente sobre qué es.

Para empezar diremos que la potencia de *XML* radica en que permite etiquetar e identificar el contenido. Se olvida, en principio, de que forma presentar el contenido. Cuando sea necesario ya le daremos un formato de salida.

Tampoco tenemos que equivocarnos y pensar que *XML* es un *HTML mejorado*. Tanto *XML* como *HTML* parten de *SGML*. Es decir, de un *metalenguaje*^s que nos permite definir lenguajes para definir la estructura y el contenido de nuestros documentos. La definición de la estructura y el contenido de un tipo de documento *SGML* se realiza en un *DTD* [**Document Type Definition**]. En él definimos los elementos que conformarán ese tipo de documentos y como tienen que estar organizados para que sea correcto.

Un ejemplo de *DTD* es el que define cómo tendrán que ser los documentos *HTML*. Por tanto, el *HTML* no es más que un tipo de documento *SGML* que se utiliza en la Web. Esta es su principal diferencia con el *XML*.

XML no es ningún tipo de documento *SGML*, sino que es una versión abreviada de éste, optimizada para su utilización en Internet. Esto significa que con él vamos a poder definir nuestros propios tipos de documentos (podremos definir nuestras propias etiquetas) y, por tanto, ya no dependeremos de un único e inflexible tipo de documento *HTML*.

^s *Metalenguaje* es un lenguaje que se usa para definir otros lenguajes.

Existen un buen número de diferencias entre la sintaxis *XML* y *HTML*. Estas constituyen las reglas básicas y es importante conocerlas para construir documentos *XML* bien formados. Estas reglas son:

- ❑ **Estructura jerárquica de los elementos:** los documentos *XML* deben seguir una estructura jerárquica correcta. Esto implica que cada etiqueta debe estar correctamente incluida en la otra. Por ejemplo, en *XML*, la siguiente línea sería incorrecta: `<B><S > HTML permite </B>esto </S>`.
- ❑ **Etiquetas vacías:** *HTML* permite usar etiquetas vacías, es decir, sin contenido. Sin embargo, en *XML*, las etiquetas vacías siempre deben cerrarse. Por ejemplo, la siguiente línea sería incorrecta en *XML*, pero no así la segunda:

`<LI> Esto es HTML <BR> en el que casi todo está permitido </LI>`

`<LI> En XML somos </BR>, más restrictivos.`

- ❑ **Un solo elemento raíz:** Los documentos *XML* sólo permiten un elemento raíz, a partir del cual todos los demás parten.
- ❑ **Valores de los atributos:** Los valores de los atributos en *XML*, al contrario que *HTML*, siempre deben aparecer encerrados entre comillas simples (‘) o dobles(“). Por ejemplo, la siguiente línea sería incorrecta en *XML*, pero no así la segunda:

`<A href=http://www.sliver.com/>`

`<A href='http://www.sliver.com'/>`

- ❑ **Tipo de letra:** *XML* es sensible al tipo de letra utilizado, es decir, trata las mayúsculas y las minúsculas como caracteres distintos. Si un elemento *XML* está definido como “LIBRO”, no podremos usar “libro” o “liBro” para referirnos a él.
- ❑ **Marcado de documentos:** En *XML* los documentos siempre deben ir precedidos de un cabecera que indique la versión de la especificación *XML* usada. Por ejemplo, en *XML* la siguiente línea sería obligatoria:

`<?xml version='1.0' ?>`

Sin embargo, *XML* también comparte algunas similitudes con *HTML* que forman parte de las reglas de formación de documentos *XML*. Ésta son básicamente dos:

- ❑ **Comentarios:** En *XML*, al igual que en *HTML*, los comentarios son contenidos entre las marcas especiales ‘<!--’ y ‘-->’. Por ejemplo:

`<!-- Esto es un comentario -->`

- ❑ **Caracteres especiales:** *XML*, al igual que *HTML*, posee un conjunto de caracteres que guardan un significado especial. Por ejemplo, el carácter **"** es usado para representar dobles comillas en el texto. Sin este carácter especial, los procesadores *XML* podrían confundir las comillas en el texto con valores de algún atributo. Los caracteres especiales en *XML* son los siguientes:

Carácter especial	Carácter que representan
&amp;	&
&lt;	<
&gt;	>
&apos;	'
&quot;	"

Como se ha mencionado, *XML* permite crear nuestra propia definición de lenguaje de marcado a través de los *DTD*. Por ejemplo, podríamos crear un *DTD* que defina una tarjeta de visita. A partir de ese *DTD*, tendríamos una serie de elementos *XML* que nos permitirían definir tarjetas de visita. En definitiva, un *DTD* es un documento de definición de tipos en el cual se describe:

- ❑ *Los nombres que se pueden emplear para los tipos de elementos.*
- ❑ *Donde pueden aparecer los distintos tipos de elementos.*
- ❑ *Como se interrelacionan los elementos.*
- ❑ *Los atributos que puede tener cada elemento.*

Veamos un ejemplo:

```
<?xml version="1.0"?>
<!ELEMENT facturas (factura+)>
<!ELEMENT factura (num_linea, cod_cliente, articulo+)>
<!ELEMENT articulo (cod_articulo, nom_articulo, cantidad, precioun)>
<!ELEMENT num_linea (#PCDATA)>
<!ELEMENT cod_cliente (#PCDATA)>
<!ELEMENT cod_articulo (#PCDATA)>
<!ELEMENT nom_articulo (#PCDATA)>
<!ELEMENT cantidad (#PCDATA)>
<!ELEMENT precioun (#PCDATA)>
<!ATTLIST factura facturaID CDATA #REQUIRED>
```

```
<facturas>
  <factura facturaID="0000">
    <num_linea>1</num_linea>
    <cod_cliente>1</cod_cliente>
    <articulo>
      <cod_articulo>1</cod_articulo>
      <nom_articulo>CD</nom_articulo>
```

```

        <cantidad>2</cantidad>
        <precioun>2200</precioun>
    </articulo>
</factura>
</facturas>

```

En este ejemplo, se ha definido un *DTD* sobre facturas. La definición del DTD permite, por ejemplo, elaborar documentos *XML* como el anterior, que son facturas de artículos. De esta forma, los DTD permiten construir ontologías primitivas y permiten ,como mencionábamos en la sección anterior, permitir comunicar entidades por motivos de interoperabilidad.

2.2 RDF

Esta sección pretende explicar qué es RDF, uno de los últimos estándares emergentes de metadatos y recomendación de *w3C* [2] para la web semántica.

La temática de esta sección se ha repartido en cuatro subsecciones para facilitar el entendimiento del lector. En la primera subsección, el lector aprenderá qué es RDF. En la segunda, se hará una breve descripción del modelo de datos. En la tercera, se hablará de qué son los vocabularios y esquemas RDF. Finalmente, se listarán las principales características de este metalenguaje en la última sección. Entonces y una vez expuesto el guión de esta sección, ya podemos empezar a ver qué es RDF.

2.2.1 Qué es RDF?

RDF significa '**Resource Description Framework**'. RDF está pensado para la web, pero dejemos esto de lado y pensemos porqué nació RDF. Para explicarlo, vamos a poner un ejemplo sencillo:

Supongamos que estamos en una biblioteca para buscar libros sobre como aprender a programar en *LISP*. En la mayoría de las bibliotecas de hoy en día usaríamos el sistema de catálogo electrónico para la búsqueda de los libros. El sistema, seguramente, nos permitiría listar los libros por *autor*, *título*, *tema* o algunos datos más. Después, tras confeccionar la lista, el sistema nos devolvería los datos de los libros etc y el dato más importante de todos: la ubicación de cada uno.

Este sistema de búsqueda, podemos decir que está basado en los que se denomina metadatos, "*datos sobre datos*", es decir, en pequeñas porciones de información (**ubicación del libro**) que describen dónde encontrar mayores volúmenes de información , en este caso los libros.

La web, en este sentido, tiene bastante que ver con una gran biblioteca. Hay muchos recursos y si conocemos la **URL** (la ubicación) podemos conseguirlos. El

problema es que en la web, actualmente, existen muy pocos metadatos. ¿Entonces como hacemos para encontrar las cosas? En general usando técnicas de fuerza bruta, donde robots buscadores como **Altavista** intentan recopilar toda la información posible y elaborar metadatos para facilitar las búsquedas. Estos sistemas operan de forma equivalente a recorrer una biblioteca, leer cada libro y almacenar las palabras clave en un catálogo.

Los webmasters y personas que han estudiado estos aspectos concuerdan en la urgencia de disponer metadatos en la web. ¿Pero como se implementarían los mismos? Si la web tuviera un todo-poderoso organizador general (**¿www.god.org?**), entonces el mismo podría establecer los criterios para describir los recursos en forma de campos (**Autor, título, tema, fecha**, etc.) Entonces simplemente esta divinidad suprema decretaría que todos las páginas deberían tener estos metadatos y eso sería todo. Por supuesto que quedarían algunos detalles por resolver, como por ejemplo, como los web sites obtendrían los metadatos, como los intercambiarían, etc.

Sin embargo, no existe ninguna organización suprema como **www.god.org**. Por esta razón, no hay forma de conseguir que todas las organizaciones responsables de generar metadatos, empiecen el mismo esquema. Si las bibliotecas que han existido por cientos de años no pueden ponerse de acuerdo en un estándar, no hay demasiadas oportunidades que las organizaciones lo hagan. Entonces, ¿Esto quiere decir que los metadatos no prosperarán y que todo el mundo construirá sus propios esquemas de datos y aplicaciones de búsqueda para siempre?.

La respuesta es no. *RDF es el esfuerzo por identificar los comunes denominadores y proveer un mecanismo para que los webmasters puedan proveer metadatos útiles para todos, sin la necesidad de ninguna intervención divina.*

2.2.2 Descripción de RDF

EL modelo de datos RDF se basa en tres tipos de objetos:

- ❑ **Recursos:** cualquier cosa que puede ser descrita usando expresiones RDF. Esto es, desde una página web hasta objetos no accesibles por la red, como por ejemplo un simple libro impreso. Los recursos se identifican mediante URIs^{**}.
- ❑ **Propiedades:** son los atributos que describen a los recursos. Cada propiedad describe los tipos de recursos que puede describir y el rango de valores que puede tomar. Ejemplo: los atributos *título* o *autor* pueden

^{**} **URI (Universal Resource Identifier)** define o especifica una entidad (y puede ser cualquier cosa: documento, dispositivo, vídeo, audio, imagen). Las **URL (Uniform Resource Locator)**, es decir, lo que todos los lectores conocerán como la dirección de una página Web (por ejemplo, **http://www.w3.org**) , es el tipo más familiar de **URI**.

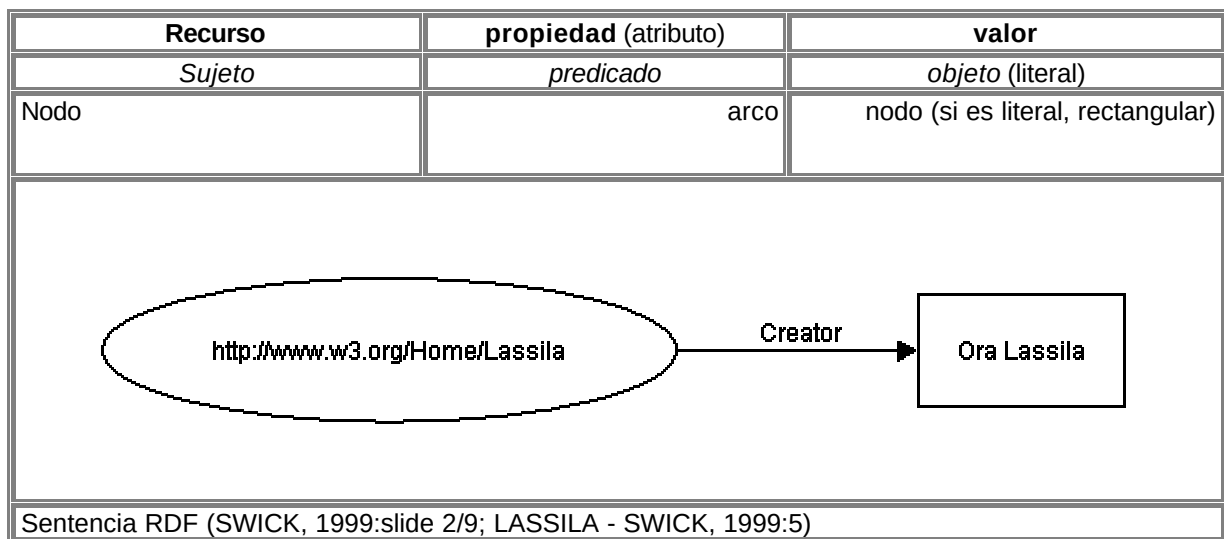
ser propiedades del recurso **libro** impreso. Las propiedades pueden tomar como valores cualquier atómico (cadena de texto, número ...) o recurso.

- ❑ **Sentencias:** ternas constituidas por un recurso, una propiedad aplicable a este recurso y el valor correspondiente a dicha propiedad. Dicho de otro modo, las sentencias RDF son simples registros de tres campos (**recurso**, **propiedad** y **valor**), con lo cuál son fáciles de manejar y de usar para buscar información en grandes volúmenes de información como la web [**escalabilidad**].

El lenguaje utilizado es **XML**, pero podrían usarse otros lenguajes.

Para ilustrar los conceptos anteriores veamos un ejemplo, con una frase sencilla:

Ora Lassila es el autor del recurso <http://www.w3.org/Home/Lassila>.



Puede leerse:

<http://www.w3.org/Home/Lasilla> tiene como autor a Ora Lasilla.

o más en general: “<sujeto> TIENE <predicado> <objeto>”.

Dicha frase estaría representada en RDF / XML de la siguiente manera:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
  <rdf:Description about="http://www.w3.org/Home/Lassila">
    <s:Creator>Ora Lassila </s:Creator>
  </rdf:Description>
</rdf:RDF>
```

La especificación formal del modelo de Febrero de 1999 define dos *sintaxis XML*: la *serializada* y la *abreviada*. La *sintaxis serializada* expresa de forma muy regular las capacidades totales del modelo. La *sintaxis abreviada* incluye términos adicionales que proporcionan una forma más compacta de representación del modelo.

La sintaxis RDF/ XML se ha diseñado para permitir *agrupar múltiples sentencias sobre un mismo recurso*. El elemento **Description** delimita esta agrupación. **Description** denomina un recurso, o bien, por medio de un atributo **about**, o bien, por medio de un atributo **ID**. Este último, es usado para subrayar que el recurso aún no existe, mientras que **about** manifiesta todo lo contrario.

Las sentencias contenidas dentro **Description** son siempre propiedades. Incluso puede haber varias propiedades con el mismo nombre (cada propiedad añade un nuevo arco al grafo). En tal caso, la interpretación de la representación gráfica se definirá por el creador del esquema.

El nombre de una propiedad puede ser cualquier cadena de texto *XML* bien formada. Además, se puede asociar a un esquema a partir de la concatenación del *namespace* del esquema RDF y el nombre de la propiedad. Más adelante, veremos que significa *namespace* y qué es un *esquema RDF*. Una propiedad puede tomar como valor un *literal* (cadena de texto, número...) o un *recurso*. Para especificar que una propiedad toma como valor otro recurso, se usa el atributo **resource**. Ejemplo:

```
<rdf:Description about="http://www.w3.org/Home/Lassila">  
  <s:Creator rdf:resource="http://www.w3.org/staffId/85740/">  
</rdf:Description>
```

Finalmente, el elemento **RDF** es simplemente un envoltorio que marca los límites del documento RDF.

2.2.3 Vocabularios y esquemas RDF

Un **vocabulario** RDF se define como el conjunto de propiedades estándares, distribuidas en paquetes para describir distintos tipos de datos, por ejemplo, para libros podría ser: autor, título y fecha.

RDF se caracteriza en sí mismo en que no posee **vocabularios predefinidos** para crear metadatos. La idea es que particulares elaboren sus propios vocabularios (puesto que no existe algo como **www.god.org**) y sólo los mejores sobrevivan y prosperen. Probablemente, los sitios web acaben adoptando un pequeño subconjunto de vocabularios que irán evolucionando con el tiempo.

La base de los **vocabularios** en RDF son los **esquemas** RDF. Estos nacieron como mecanismo para asegurar que tanto lector como escritor de una sentencia [**declaración**] entendiera el mismo significado, sin ambigüedad, para los términos

utilizados, tales como **Creator**. Puesto que especialmente, en el ámbito global del World Wide Web conviene ser lo más preciso posible, el contenido de los esquemas RDF sirve para expresar el significado de los términos utilizados.

Un **esquema** RDF podemos considerar, a su vez, que es como una especie de *diccionario*. Es decir, un diccionario que define todos aquellos términos que se utilizarán en una sentencia RDF común y que servirán, a su vez, para dotarla de significado específico.

Sin embargo, un vocabulario RDF no es un conjunto de esquemas RDF, compuestos de *elementos*, *valores* de estos elementos y sus relaciones *semánticas*. Es más bien, un conjunto de recursos que evoluciona a través del tiempo. Son los esquemas RDF los que designan versiones particulares de un vocabulario RDF en cualquier momento puntual. Ello nos indica como RDF permite implementar el concepto de evolución del *Web Semántico*.

En un esquema RDF se documentan las *definiciones* y *restricciones* de uso de las propiedades. Para evitar confusiones entre definiciones independientes del mismo término (en otros esquemas), RDF utiliza los **namespaces** de XML. Los namespaces [“*espacios de nombre*”] son una forma de asociar el uso específico de una palabra en el contexto de una esquema [*diccionario*] en el que se puede encontrar una definición determinada. Es la forma de identificar el uso de una palabra dentro de un determinado contexto. Es decir, si dos esquemas definiesen el término **Creator** con significados distintos, el uso de esta propiedad en una sentencia podría llegar a generar ambigüedad. Para solucionar este evento, se asocia el *namespace* de cada esquema (el w3c declara que cada esquema tiene su *namespace* propio) a cada propiedad y evitar así confusiones. Los **namespaces** se expresan en RDF mediante el atributo **xmlns**.

En definitiva los **namespaces** proporcionan un mecanismo que permite:

- ❑ Que las referencias al nombre de una propiedad no sean ambiguas.
- ❑ Que los documentos RDF puedan tener información definida por propiedades de distintos esquemas.

Los esquemas RDF proporcionan mecanismos para establecer las relaciones que hay entre los distintos recursos, así como sus definiciones. Para empezar, describiremos el sistema de tipos de RDF.

En RDF los recursos pueden ser instancias de una o más **clases**. Las **clases** se organizan en forma de jerarquías de clases, en las cuales por ejemplo, ‘*animal*’ es la superclase de ‘*persona*’. El concepto de clase se corresponde al concepto genérico de tipo o categoría, equivalente al concepto de clase de programación orientada a objetos. Para expresar las relaciones que hay entre las subclases, en RDF, se utiliza la propiedad **rdfs:subClassOf**.

La propiedad ***rdf:type*** se usa para indicar si un recurso es una clase [***rdfs:Class***], una propiedad [***rdf:Property***] o simplemente un recurso [***rdfs:Resource***]. La reunión de ***rdfs:Class***, ***rdf:Property*** y ***rdfs:Resource*** constituye el corazón de la especificación del RDF esquema.

Para entender estos conceptos, veamos un ejemplo:

```
<rdf:RDF xml:lang="en"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

  <rdf:Description rdf:ID="Animal">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class" />
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Resource" />
  </rdf:Description>

  <rdf:Description rdf:ID="Person">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class" />
    <rdfs:subClassOf rdf:resource="#Person" />
  </rdf:Description>

</rdf:RDF>
```

Los esquemas RDF proporcionan mecanismos para establecer restricciones en el uso de los predicados. La característica (propiedad) “es un padre”, por ejemplo, sólo se aplica a personas. En particular, los conceptos de dominio y rango son los conceptos usados en los esquemas RDF para crear sentencias que tengan sentido en el mundo real. Un esquema que viole alguna restricción adquiere la categoría de ***inconsistente***.

La restricción de rango se expresa mediante la propiedad ***rdfs:range***. Esta propiedad delimita el tipo de valores que puede recibir una propiedad. Por ejemplo, la propiedad ‘***Creator***’ únicamente puede recibir como valor un recurso de la clase ‘***Person***’.

La restricción de dominio se expresa mediante la propiedad ***rdfs:domain***. Esta propiedad se utiliza para determinar sobre que clases de recursos se puede aplicar una propiedad para que tenga sentido. Por ejemplo, la propiedad ‘***Title***’ se espera que se puede aplicar sobre la clase ‘***Book***’ y no sobre ‘***Person***’.

Para entender estos conceptos, veamos el siguiente ejemplo:

```
<rdf:RDF xml:lang="en"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

  <rdf:Description rdf:ID="Title">
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Property" />
```

```

    <rdfs:domain rdfs:resource="#Book"/>
    <rdfs:range rdfs:resource="http://www.w3.org/datatypes#String"/>
</rdf:Description>

<rdf:Description rdf:ID="Height">
    <rdfs:type rdfs:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Property"/>
    <rdfs:domain rdfs:resource="#Person"/>
    <rdfs:domain rdfs:resource="#Tree"/>
    <rdfs:range
rdf:resource="http://www.w3.org/datatypes#Number"/>
</rdf:Description>
</rdf:RDF>

```

La ventaja más importante de RDF comparado con otros enfoques de metadatos es que usando RDF, puede decirse cualquier cosa sobre cualquier cosa. Cualquiera puede hacer declaraciones RDF sobre cualquier recurso identificable. Utilizando RDF, los problemas de ampliar metadatos y combinar metadatos de diferentes esquemas desaparece, pues RDF no utiliza documentos cerrados.

2.2.4 Características de RDF

RDF esta cuidadosamente diseñado para tener las siguientes características:

- ❑ **Independencia:** Dado que una propiedad es un recurso, toda organización independiente o incluso cada persona puede inventarlas. Podemos inventar una propiedad llamada "**Autor**" y otros pueden inventar una propiedad llamada "**Director**" que podría aplicarse, por ejemplo, a recursos asociados con películas. Esta libertad es necesaria porque no hay un dios que se encargue de definir cuáles son las propiedades a usar en todo el mundo
- ❑ **Intercambio:** Dado que las sentencias RDF se escriben en XML pueden ser fácilmente usadas para intercambiar información. Esto eventualmente es necesario aun con la existencia de un dios.
- ❑ **Escalabilidad:** Las sentencias RDF son simples registros con tres campos (Recurso, propiedad, valor), por lo que son fáciles de manejar y de usar para buscar objetos aun en volúmenes realmente grandes. La web ya es lo suficientemente grande y continua creciendo. Es probable que tengamos en algún momento miles de millones de RDFs flotando a nuestro alrededor algún día. Por eso la escalabilidad es importante.

3. Agentes y Sistemas MultiAgentes

Antes de empezar a definir qué son los sistemas MultiAgentes, cualquiera de nosotros se preguntará, en primera instancia, ¿Qué es un agente?. La respuesta no es nada trivial. La comunidad científica, en la actualidad, aún no se ha consensuado sobre una definición estándar. De momento, cada organismo ha tendido a definir los agentes según sus criterios. Así pues, para empezar este capítulo, nada mejor podemos hacer que dar varias definiciones sobre qué es un agente y explicar cuáles son los principales tipos.

3.1 Definición de Agente

Esta sección pretende explicar qué es un agente. Como la comunidad científica aún no ha determinado de manera unívoca que se entiende por agente, proponemos unas cuantas definiciones para explicar este concepto. La idea es intentar reflejar los distintos puntos de vista sin ligarse a una definición concreta. Estas definiciones han sido extraídas del PFC de **David Isern Alarcón**[3].

Nicholas Negroponte y **Alan Kay** [4], enuncian:

Un agente puede ser considerado un asistente personal software [**personal software assistant**] con autorización delegada por el usuario. El objetivo de este tipo de software es realizar tareas que serían arduas y complicadas de realizar por el usuario. El funcionamiento de estos programas se basaría en simular las relaciones humanas y efectuar operaciones que nadie más podría realizar por ti.

Para **Hyacinth S. Nwana** de BT y **Michael Wooldridge** de la Universidad de Liverpool[5]:

Un agente es un componente de software y/o hardware capaz de realizar tareas en representación de un usuario. Que tareas realizar y cuando, son parámetros que se han de proporcionar en forma de expresiones, metaexpresiones o clases. Esta información define los diferentes tipos de agentes y es necesario escoger de que tipo se trata.

Según **Pattie Maes** del MIT [6]:

Un agente es un sistema computacional que habita en un entorno complejo y dinámico. El agente puede sentir y actuar de forma autónoma en su entorno, y tiene un conjunto de objetivos o motivaciones potencialmente alcanzables a través de sus acciones.

Para la **Foundation for Intelligent Physical Agents (FIPA)**[7]:

Un agente es una entidad que reside en un entorno, donde interpreta datos de sensores reflejando eventos del entorno, y ejecuta comandos que provocan cambios en éste. Un agente puede ser software o hardware. En este último caso, se necesita de mucho software para hacer al agente hardware.

Como se puede extraer de las definiciones anteriores, el concepto de agente es un concepto *heterogéneo*. No existe una definición estándar que precise de forma única que es un agente. Incluso en la mayoría de las definiciones anteriores, ni tan sólo hay una coincidencia en los términos empleados. Todo esto hace que exista una gran confusión a la hora de determinar que entidades se pueden considerar agentes. Esto es, entidades que son consideradas por ciertos investigadores agentes, pueden no serlo para otros científicos. De esta forma, y delante la imposibilidad de que la comunidad científica encuentre una definición global, los agentes habitualmente se definen a partir de un conjunto de atributos o propiedades. Es decir, aquellas propiedades que convierten un sistema hardware y/o software corriente en un agente.

3.1.1 Atributos

Esta sección pretende listar cuáles son las propiedades que aplicadas a un sistema *hardware* y/o *software* permiten considerarlo un agente. Evidentemente, un agente no tiene porque poseerlas todas. Para ciertos científicos, un agente puede ser la reunión de unas propiedades que para otros no sean suficientes. Por tanto, estos atributos sólo señalan cuáles son las características que puede poseer un agente, nada más. Éstas son:

□ Autonomía

Los agentes han de actuar sin la intervención directa de los humanos. Además han de tener algún tipo de control sobre las sus acciones y estado interno. Esto requiere que el agente sea capaz de realizar acciones periódicas y ejecuciones espontáneas, tanto preventivas como independientes, y de tomar iniciativas en pro del usuario.

□ Sociabilidad / Cooperación

Es la capacidad de interacción de una agente con respecto a los demás o con un humano expresada por medio de algún lenguaje de comunicación. La interacción entre usuario y agente puede ser descrita como un contrato de colaboración, donde el usuario proporciona unas especificaciones y el agente su capacidad de trabajo. Por ejemplo, una petición de descarga de un fichero a una agente encargado de gestionar un servidor *FTP*^{††}. Sin embargo, la interacción entre agentes es constituida por el mecanismo a través del cual los agentes intercambian su conocimiento, sus creencias y sus planes de trabajo colectivos, para solucionar problemas de mayor complejidad.

^{††} **FTP**: acrónimo de File Transfer Protocol, es un protocolo estándar para el intercambio de fichero entre máquinas remotas vía Internet.

❑ ***Reactividad***

Es la capacidad de reacción de un agente delante de los cambios en el entorno. Las acciones se realizan cuando se cumplen las precondiciones de las reglas o por ejecución de planes prediseñados.

❑ ***Proactividad***

Es la capacidad de los agentes de tomar la iniciativa en pro del usuario. Es decir, los agentes han de ser capaces de exhibir algún tipo de comportamiento que les lleve a alcanzar sus metas sin esperar estímulos del entorno y por propia iniciativa.

❑ ***Movilidad***

Es la propiedad de todos aquellos agentes que pueden desplazarse a través de la red. Así pues, pueden ejecutar sus tareas de forma remota y mejorar la eficiencia. Por ejemplo, un agente podría gestionar una base de datos de forma remota, pero si el agente fuera capaz de desplazarse al servidor de base de datos con todas las peticiones, podría ejecutar sus tareas *in situ* y volver con los resultados. Entonces, el número de transmisiones se habría reducido a dos (ida y vuelta del agente) a parte de liberar más tiempo el canal. Eso mejoraría la eficiencia.

❑ ***Continuidad temporal***

Esta propiedad se refiere a que los agentes ejecutan los procesos de forma continuada. No ejecutan una tarea y finalizan.

❑ ***Racionalidad***

Es la capacidad de los agentes de actuar de forma racional para alcanzar sus objetivos. Es decir, un agente siempre se limitará a realizar sus tareas, únicamente, cuando las condiciones sean satisfechas.

❑ ***Veracidad***

Es la asunción de que los agentes no transmiten información falsa de forma premeditada.

❑ ***Bondad***

Es la asunción de que los agentes no tienen intereses conflictivos, y, que siempre estarán disponibles para realizar aquellas tareas que les sean requeridas.

3.1.2 Tipos de agentes

Diferentes investigadores han intentado de realizar clasificaciones sobre los tipos de agentes. En general, las clasificaciones propuestas, forman grupos bastante genéricos y no impiden que la mayoría de las tipologías aparezcan mezcladas en los programas reales. A continuación, veremos las características fundamentales de cada uno de ellos:

- ❑ **Agentes de Internet / Información:** son aquellos agentes que tienen acceso a fuentes de información [**Internet**] y son capaces de responder a las peticiones del usuario y/o otros agentes manipulando la información según un cierto orden. Muchas descripciones de agentes de Internet, tienden a definir estos agentes como entidades que vagan por Internet recogiendo información. En realidad, estos tipos de agentes son realmente difíciles de diseñar debido al gran volumen de información disponible. Además, sería necesario diseñar una interfaz que permitiera al usuario indicar la información a extraer y una vez devuelta permitiera su interpretación.
- ❑ **Agentes de Interfaz:** son agentes que colaboran con el usuario para resolver ciertas tareas complicadas. Este tipo de agentes observan y monitorizan las acciones tomadas por el usuario, aprendiendo de éstas, en vistas a proponer mejores formas de llevar a cabo ciertas tareas.
- ❑ **Agentes de Colaboración:** son agentes con gran capacidad de sociabilidad o cooperación con otros agentes. Estos agentes usan esta capacidad, para resolver tareas de gran envergadura, no factibles a nivel individual. Por tanto, son agentes que requieren gran capacidad de comunicación y de una metodología que les permita ser capaces de entenderse entre sí. La aparición de mensajes **ACL** [**Agent Communication Language**] ha sido pensada para salvar este último requisito. Además, sería necesario el uso de protocolos de transmisión de datos tales como **TCP / IP^{‡‡}**.
- ❑ **Agentes móviles:** estos agentes tienen como mayor atractivo su capacidad para desplazarse por la red. Esta característica, les permite extraer y manipular la información **in situ** y, mejorar la eficiencia. Para que un agente pueda desplazarse, es necesario que éstos sean capaces de interactuar con los **hosts** remotos.
- ❑ **Agentes reactivos:** son agentes que tienden a actuar a partir de estímulos del entorno y no a partir de modelos simbólicos de representación. El hecho de reaccionar ante estímulos a partir de

^{‡‡} **TCP/ IP:** es un modelo de referencia que surge de la necesidad de interoperar redes diversas. Se compone de 4 capas: Host-Red, Internet, Transporte, Aplicación.

patrones, hace que estos agentes sean simples al igual que sus interacciones con otros agentes.

- **Agentes híbridos:** estos agentes están constituidos por una mezcla de otros tipos en vistas a minimizar las deficiencias, de usar otras tipologías por separado. Por ejemplo, un agente podría necesitar disponer de una gran capacidad de adaptación al entorno, y simultáneamente, de grandes dosis de planificación. Entonces, este agente debería ser una combinación de agente reactivo y deliberativo.

3.2 Arquitectura de SMA

Una vez comentado qué se puede entender por agente, esta sección, pretende dar una definición de qué es un *SMA* y cuáles son los elementos constituyentes básicos. Para empezar definiremos qué es un *SMA*:

<< Un **Sistema MultiAgente** o **SMA** es un sistema constituido por las interacciones de los diversos agentes constituyentes>> .

Entonces, los **SMA** no son más que la reunión de un conjunto de agentes en vistas a realizar tareas más complejas a través de sus interacciones. Sin embargo, los **SMA** no se pueden construir simplemente mediante la suma de agentes. Es necesario fijar una normativa de diseño e implementación. Esto es expuesto a continuación:

Una de las funciones de la **Foundation for Intelligent Physical Agents (FIPA)**[7] es proporcionar una serie de normas para implementar y diseñar SMA de forma correcta. Debido a la gran diversidad de SMA, la *FIPA* proporciona un estándar que constituye un **marco de referencia** para los diferentes sistemas. Esto es, no siempre las especificaciones de la *FIPA* son la mejor solución para el diseño de un SMA, pero si constituyen un referente sobre el cual basar un diseño. Además, estas especificaciones ofrecen gran versatilidad y permiten en la mayoría de los casos adaptarse a gran variedad de implementaciones.

La administración de agentes facilita una normativa de trabajo dentro de la cual los agentes existen y operan. Esta normativa puede ser encontrada dentro de la **FIPA 98 Specification**[8]. Ésta establece un modelo lógico para la creación, registro, comunicación, movilidad y destrucción de agentes.

En la siguiente figura, podemos ver todos los elementos de la arquitectura propuestos por la *FIPA*:

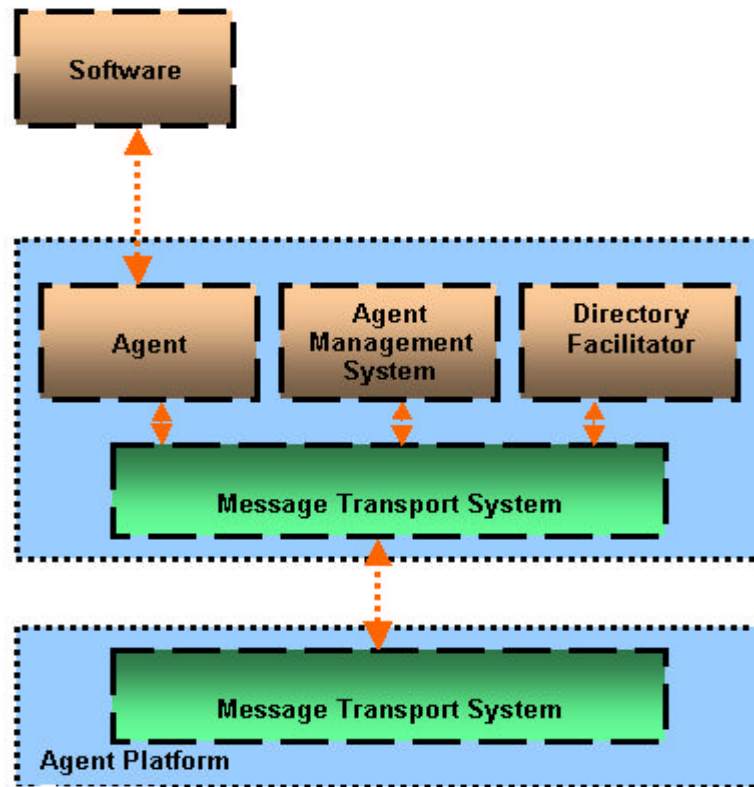


Figura 4. Arquitectura propuesta por la FIPA para los SMA.

A continuación, comentaremos cada uno de los componentes lógicos propuestos por la FIPA:

- ❑ **Agents:** son los agentes constituyentes del **SMA**. Soportan varios tipos de identificación. El más utilizado es el identificador único global o **GUID**. Pueden tener más de un propietario.
- ❑ **Agent Platform(AP):** proporciona la infraestructura física donde los agentes pueden ser ubicados. La **AP** está constituida por uno o varios computadores, un sistema operativo, software de soporte a agentes, los componentes de administración de la plataforma según el estándar **FIPA (DF, AMS, ACC)** y además, de una plataforma interna responsable del transporte de mensajes(**IPMT**).
- ❑ **Directory Facilitator(DF):** es un agente cuya función es proporcionar un servicio de **páginas amarillas** entre los agentes de la plataforma. Es decir, los agentes pueden registrar sus servicios o consultar cuáles son ofrecidos por los demás agentes a través del **DF**.
- ❑ **Agent Management System(AMS):** es el agente supervisor de acceso y uso de la plataforma. Sólo un **AMS** puede existir dentro de una **Agent Platform(AP)**. El **AMS** mantiene un directorio con los nombres lógicos de los agentes y sus direcciones de transporte, ligadas, a la plataforma

que gobierna. También, ofrece un servicio de **páginas blancas** a los demás agentes.

- ❑ **Agent Communication Chanel(ACC):** cualquier agente de la plataforma tiene acceso a al menos un **ACC**. Constituye el método de comunicación por defecto entre agentes de diferentes plataformas o **AP**.
- ❑ **Internal Platform Message Transport(IPMT):** es el propietario del medio de intercambio de mensajes dentro de una **AP**.

3.2.1 DIRECTORY FACILITATOR(DF)

El agente DF proporciona un servicio de **páginas amarillas** entre los agentes. Es un agente obligatorio a quien es confiada la custodia del directorio de agentes. Su objetivo es mantener una lista completa, con información actualizada, de los servicios ofrecidos por los agentes dentro de la plataforma. En principio, no hay un número fijo de agentes DF dentro de una AP. Sin embargo, es obligatorio contener a un DF dentro de una plataforma como mínimo. Un agente DF, entre otras propiedades, tiene la capacidad de restringir el acceso a la información contenida en su directorio. Además, puede realizar comprobaciones de acceso a los agentes que intenten informar sobre los cambios en su estado. DF no controla el ciclo de vida de ningún agente dentro de la plataforma.

Los agentes pueden registrar sus servicios en DF. También, pueden preguntarle que servicios son ofrecidos por otros agentes. Además, y de manera similar, el AMS y ACC también pueden registrarse a un DF. Incluso un DF puede registrarse en otros DF.

El agente DF está constituido por un mecanismo de búsqueda, que en primera instancia, es local, con posibilidad de ampliar la búsqueda a otros DF si es necesario. Para propósitos más específicos se pueden utilizar las siguientes restricciones:

- ❑ En el número de respuestas: **“df-search-resp-req”**.
- ❑ En el número de saltos: **“df-search-depth”**.
- ❑ En el tiempo máximo: **“df-search-time-out”**.
- ❑ En el protocolo de búsqueda: **“df-search-algo”**.

Las acciones de administración soportadas por DF son:

- ❑ Register
- ❑ Search
- ❑ Deregister
- ❑ Modify

3.2.2 AGENT MANAGEMENT SYSTEM(AMS)

El **AMS** es un componente obligatorio y único dentro de una AP. Su tarea principal es gestionar las operaciones. Por tanto, es responsable de creación y destrucción de los agentes, de decidir si un agente puede registrarse de forma dinámica a una plataforma e incluso de supervisar las migraciones de agentes desde una plataforma hacia otra.

El AMS supone la autoridad administrativa de una AP. Así pues, AMS tiene privilegios para pedir a una agente que finalice su ejecución o en caso contrario, forzar su terminación, por ejemplo. AMS mantiene un índice de todos los agentes residentes en una plataforma. Este índice incluye: el **GUID** [*Global Unique Identifier*] y la dirección de transporte asociada a la AP, por cada agente. Un agente es considerado residente de una plataforma si ha sido registrado por el **AMS** de ésta.

Las acciones obligatorias de administración soportadas por todo **AMS** son:

- ☐ Register-agent
- ☐ Deregister-agent
- ☐ Modify-agent
- ☐ Query-platform-profile
- ☐ Authenticate
- ☐ Search-agent

Otras operaciones no obligatorias que puede soportar son:

- ☐ Suspend-agent
- ☐ Terminate-agent
- ☐ Create-agent
- ☐ Resume-agent-execution
- ☐ Invoke-agent
- ☐ Execute-agent
- ☐ Resource-management

3.2.3 AGENT COMMUNICATION CHANNEL(ACC)

ACC es el componente encargado de encaminar los mensajes de agentes de una plataforma hacia agentes de otras plataformas. La FIPA garantiza que todos los agentes han de disponer de como mínimo un ACC. Sólo los mensajes para agentes pueden ser enviados a un ACC.

ACC facilita el enrutamiento de los mensajes. Este enrutamiento requiere de un acuerdo mutuo entre las diferentes plataformas, sobre que protocolo de transporte, codificación y esquema de direccionamiento usar. En contraste, un agente registrado en una plataforma, dispone, al instante, de un mecanismo para el intercambio de mensajes entre agentes de la misma.

El ACC es, en definitiva, un agente que controla la comunicación entre plataformas. Además, posee una interconexión con el mecanismo de IPMT que FIPA no define.

3.2.4 AGENT PLATFORM(AP)

Los agentes **FIPA** están ubicados físicamente en una **AP** y utilizan las facilidades ofrecidas por ésta, para desarrollar sus tareas o funciones. En este contexto, un agente, se convierte en un proceso físico que tiene un ciclo de vida que es gestionado por la **AP**.

El ciclo de vida de un agente FIPA posee las siguientes características:

- ❑ **Circunscrito a una plataforma:** un agente sólo es gestionado dentro de una AP.
- ❑ **Independiente de la aplicación:** el ciclo de vida de un agente es independiente de cualquier aplicación del sistema. Este ciclo está definido sólo por estados y sus transiciones.
- ❑ **Orientado a la instancia:** un agente que posea el ciclo de vida descrito es considerado una instancia. Es decir, un agente con nombre propio y ejecución independiente.
- ❑ **Singularidad:** cualquier agente en cualquier instante, siempre está descrito por su estado en el ciclo de vida y por su **AP**.

Las acciones que se pueden efectuar sobre los agentes son:

- ❑ **Create:** creación de un agente
- ❑ **Invoke:** invocación de un agente nuevo.
- ❑ **Destroy:** terminación forzada de un agente. Esta acción sólo puede ser iniciada por el AMS. Además, nunca puede ser ignorada por el agente receptor.
- ❑ **Quit:** terminación normal de un agente.
- ❑ **Suspend:** suspende la actividad de un agente. Puede ser iniciada por cualquier agente o por el AMS.
- ❑ **Resume:** coloca el agente de estado de suspendido al estado activo. Sólo el AMS puede ser el iniciador de esta acción.
- ❑ **Wait:** coloca el agente en estado de espera. Esta acción sólo puede ser iniciada por el propio agente.

- ❑ **Wake Up:** devuelve el estado de activo al agente, después de estar en estado de espera. Esta acción al igual que Resume, sólo puede ser iniciada por AMS.

Las acciones siguientes sólo son usada por agentes móviles:

- ❑ **Move:** el agente está en estado transitorio. Sólo puede ser iniciado por el propio agente.
- ❑ **Execute:** coloca el agente del estado transitorio al estado activo. Sólo puede realizar esta acción el AMS.

3.3 Lenguaje de Comunicación

Esta sección pretende explicar qué son los lenguajes de comunicación entre agentes y cuáles son los protocolos básicos de comunicación. De esta forma, y siguiendo el orden anterior, esta sección se ha dividido en dos partes. En la primera, se detalla qué es un lenguaje de comunicación. En la segunda, enmarcada en una *subsección*, se explicará cuáles son los protocolos básicos de la comunicación *interagente*. Así pues, empecemos por la primera parte:

Los agentes de forma individual pueden alcanzar un buen nivel de funcionalidad, no obstante, su funcionalidad puede aumentar cuando los agentes cooperan. De esta manera, es factible resolver ciertos problemas que de forma individual, serían difíciles de manejar por su complejidad. Además, el conocimiento de cada agente es limitado, incluso puede ser incompleto o incorrecto, y participar en un colectivo puede beneficiar a todos. Para poder cooperar, los agentes se han de comunicar los unos con los otros. Por tanto, es necesario disponer de un lenguaje de comunicación para agentes o **ACL** [**Agent Communication Language**].

Los lenguajes **ACL** están fundamentados sobre la idea de que las comunicaciones individuales entre agentes, han de reducirse a un pequeño grupo de ideas o más generalmente, actos comunicativos. Estos actos aportan el significado básico a la comunicación. A continuación, mostramos un ejemplo de mensaje **ACL**:

```
(INFORM
  :sender      a1@localhost
  :receiver    a2@localhost
  :content
              (get (in hpl-server) acces)
  :reply-with  hpl01
  :language    sl
  :ontology    hpl-ontology
)
```

Los mensajes **ACL** constituyen el elemento básico de la comunicación. Son representados mediante expresiones. La FIPA es quien determina cuál es el formato de los mensajes y su uso. Así pues, cada mensaje tiene una estructura prefijada formada por parámetros o *slots*. El uso de los *slots*, estará determinado por nuestras necesidades. El primer elemento de un mensaje es la palabra que identifica el acto comunicativo, el cual, define el significado del mensaje. En la tabla 1, hay una lista de los actos comunicativos soportados por la FIPA y su significado. A continuación, le siguen los *slots*, precedidos por el carácter ‘:’ e identificados por una palabra clave. Estos *slots* son básicamente:

□ **:content**

Este parámetro contiene almacenado el contenido del mensaje. Este contenido aparece codificado en forma de expresión por algún lenguaje de representación.

□ **:sender** y **:receiver**

Indican cuál es el emisor y receptor/es del mensaje. Estos slots ayudan a determinar dónde se ha de entregar el mensaje y a dónde replicar.

□ **:language** y **:ontology**

Sirven para determinar en que lenguaje se ha codificado y cuál es la ontología que da significado al campo **:content**, respectivamente. Algunos de los valores del slot **:language** pueden ser: *LISP*, *JAVA*, *FIPA-SL*, *FIPA-RDF* etc. Este campo es utilizado por los agentes, para determinar si la información de los mensajes está escrita en una lengua conocida y por tanto, es analizable.

El slot **:ontology** permite determinar a los agentes de que se habla. De esta forma, los agentes podrán interpretar los términos de **:content** de manera inequívoca y entender su significado.

□ **:reply-with** y **:reply-by**

Estos slots permiten a los agentes controlar el estado de la comunicación. El slot **:reply-with** contiene un identificador que sirve para indicar en que estado del acto comunicativo se encuentran los agentes. Mientras que el campo **:reply-by**, consiste en una marca de **timeout**. De esta forma, los mensajes pueden disponer de una fecha de caducidad. Esto es, de una fecha que una vez superada, permite optar a los agentes por ignorar el mensaje.

accept-proposal	Aceptación de una propuesta para realizar una acción.
agree	Conformidad para realizar una acción.
cancel	Cancelación de alguna petición anterior aún no realizada.
cfp	(Call For Proposal) Petición de propuestas a un conjunto no vacío de agentes en vistas a realizar un acción.
confirm	Confirmación de que una proposición es cierta.
disconfirm	Confirmación de que una proposición es falsa cuando se consideraba cierta.
failure	Acción de informar al receptor de algún problema durante la tarea encomendada.
inform	Acción de informar al receptor de algún hecho.
not-understood	Acción de informar al receptor de que no se ha entendido el mensaje previo.
propose	Realización de una propuesta en respuesta a un CFP previo.
query-if	Acción de preguntar al receptor sobre una proposición.
refuse	Rechazo de alguna petición.
reject-proposal	Rechazo de una propuesta durante una negociación o CFP.
request	Petición al receptor de realizar una acción.
request-when	Petición al receptor de realizar una acción cuando se cumpla alguna proposición.

Tabla1. Actos comunicativos permitidos por la FIPA.

3.3.1 PROTOCOLOS DE COMUNICACIÓN

De manera similar a los humanos, los agentes siguen patrones típicos en sus conversaciones. Esto es, a cada paso del acto comunicativo, se espera que los agentes contesten con un tipo concreto de mensaje. Estos patrones prefijados se denominan **protocolos** y sirven para que los agentes mantengan las conversaciones de forma correcta.

Los principales protocolos de comunicación son:

1. FIPA Request

Cuando una agente **initiator** inicia este protocolo hacia un agente **participante** desea que el agente receptor realice alguna tarea. El agente initiator envía un mensaje del tipo **request** al agente participante, en primera instancia. Este agente recibe la petición, la estudia e investiga si puede realizar la acción propuesta. En caso afirmativo, replica con un mensaje **agree**. En caso negativo, replica con un mensaje **refuse**. Si el initiator no ha codificado correctamente el mensaje y el participante no es capaz de descifrar su contenido, este último contesta con un mensaje **not-understood**. Entonces, el agente participante procede a realizar la acción. Si la acción una vez finalizada llega a buen término, el agente participante contesta con un **inform**, caso contrario, con un mensaje **failure**.

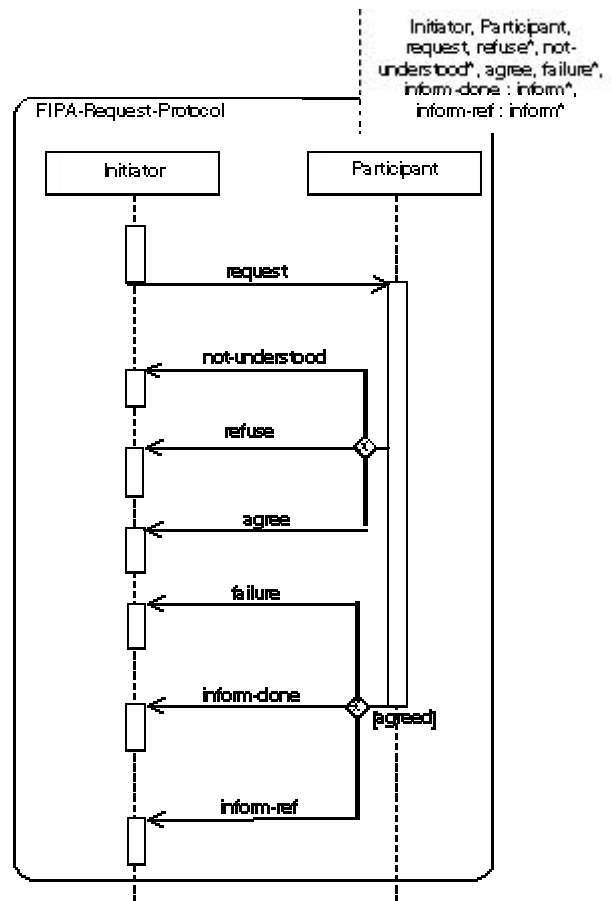


Figura 5. Protocolo de interacción FIPA Request

2. FIPA Query

Cuando una agente **initiator** inicia este protocolo hacia un agente **participante** desea preguntar al agente alguna proposición. El agente

initiator envía un mensaje del tipo **query** al agente participante, en primera instancia. Este agente recibe la petición, se da cuenta de que es una pregunta y la estudia. Finalmente, el participante contesta con alguna de las cuatro posibilidades. Contesta con una **inform**, si es capaz de obtener una respuesta. Contesta con un **refuse**, si no acepta la pregunta o con un **failure**, si no es capaz de obtener una repuesta (error de conexión a una base de datos, por ejemplo). También, puede contestar con un **not-understood** si el initiator no ha codificado el mensaje de manera correcta.

De mensajes de tipo **query** hay de dos tipos: los **query-if** y los **query-ref**. La única diferencia entre los dos tipos, es que los mensajes del primer tipo, reclaman una contestación afirmativa o negativa sobre una proposición, mientras que los del segundo tipo reclaman una respuesta en forma de expresión.

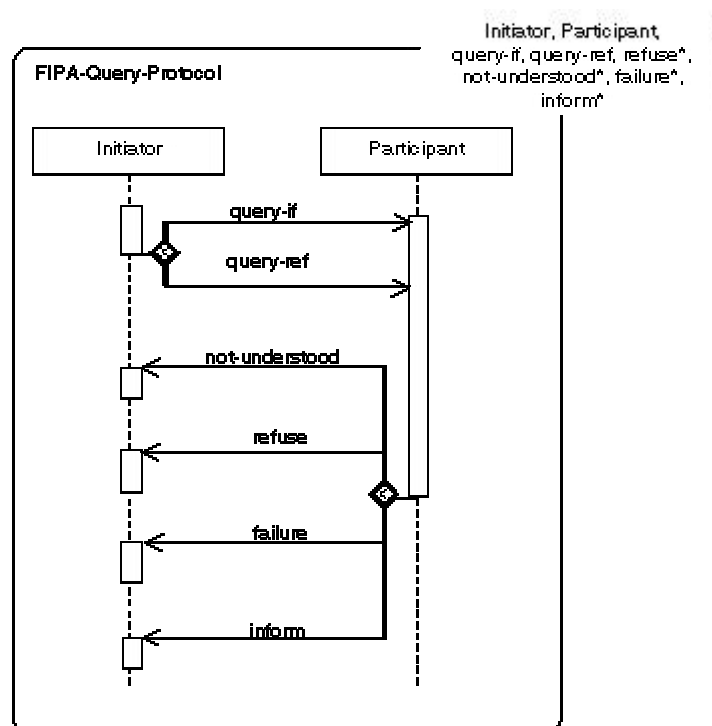


Figura 6. Protocolo de interacción FIPA Query

3. FIPA Contract Net

FIPA Contract Net es el protocolo de mayor complejidad y duración de los protocolos anteriores. Como en todos los protocolos anteriores, también hay un agente **initiator** y una agente **participante**. El objetivo de este protocolo es permitir a los agentes **participantes** proponer propuestas sobre una determinada petición del agente **initiator**

y realizar alguna si este último acepta. Los pasos del protocolo son los siguientes:

El agente **initiator** envía un mensaje del tipo **call for proposal** o **cfp** al agente participante, en primera instancia. Entonces, el agente participante recibe la petición, la estudia y si puede envía una propuesta al agente **initiator**. También, puede rechazar la petición mediante un mensaje **refuse**. En este instante, el agente **initiator** estudia la propuesta. Si la propuesta le parece bien, manda un **accept-proposal** al agente participante, o en caso contrario, manda un mensaje del tipo **reject-proposal**. Si la petición del agente participante es aceptada, el agente procede a realizar la acción o contesta con un **cancel** para cancelar su propuesta. Finalmente, si la acción es realizada envía un **inform**, sino, envía un **failure**.

Este protocolo, en el caso más general, está pensado para ser una comunicación de **1:N**. En este caso, el agente **initiator** debe filtrar y escoger un subconjunto de las propuestas.

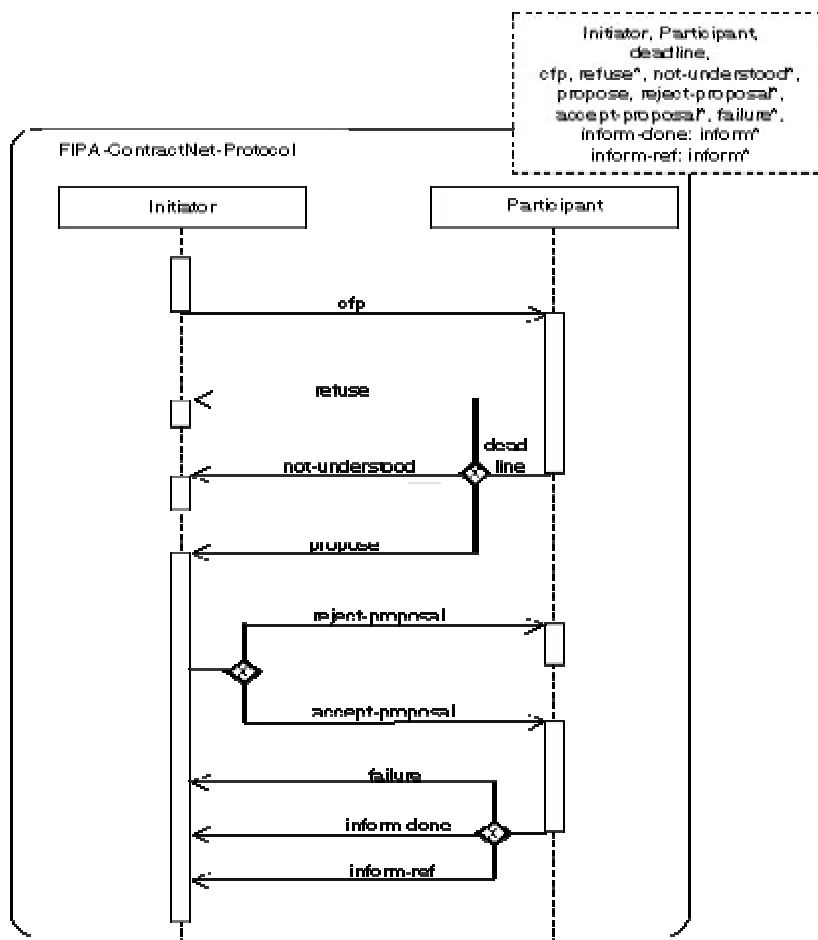


Figura 7. Protocolo de interacción FIPA Contract Net

4. JADE: Entorno de desarrollo de SMA

El capítulo tercero de esta documentación, como el título sugiere, pretende explicar la herramienta de desarrollo de **SMA** escogida. Este capítulo, abarca en su contenido, todos los aspectos fundamentales en el desarrollo de cualquier **SMA**. Esto es, desde como se define un agente hasta como se comunican entre ellos. A cada una de estas facetas se le ha dedicado una sección. Así, por ejemplo, la **sección 4.1** se dedicará a explicar como se definen los agentes usando esta herramienta.

Para empezar este capítulo, nada mejor podemos hacer que explicar cuál es la herramienta elegida. Esto nos ocupará las siguientes líneas. Además, al final de esta introducción, también explicaremos el motivo de nuestra elección. Entonces, empecemos definiendo ya qué es **JADE**:

JADE [**Java Agent Development Framework**] es un entorno *software* orientado a desarrollar sistemas Multiagentes y aplicaciones de agentes en general, siguiendo las especificaciones estándares sobre agentes inteligentes de la FIPA.

JADE, como hemos mencionada anteriormente, es un entorno y como tal, no soluciona la totalidad de los problemas que puedan surgir en el desarrollo de cualquier SMA. Es más bien, un conjunto de librerías que solucionan problemas de bajo nivel, que aparecen siempre en cualquier diseño con agentes.

JADE está escrito en lenguaje JAVA, y por tanto, los agentes también serán escritos en JAVA. La elección de este lenguaje está determinada por sus características, tales como su paradigma *POO*^{§§} o su capacidad para desarrollar aplicaciones de objetos distribuidos. Características fundamentales, al desarrollar cualquier SMA.

JADE sigue la propuesta de la FIPA sobre el diseño de una arquitectura SMA: los agentes residen en un entorno denominado plataforma o AP. Cada plataforma está dividida a su vez, en contenedores que contienen agentes. Podemos identificar cada contenedor con un dominio de agentes. Así pues, podríamos pensar que cada contenedor es una máquina que contiene agentes que extraen información de Internet.

La plataforma **JADE** proporciona, además, un entorno seguro que permite que los agentes entren en contacto e interactúen entre sí.

Tal como define la FIPA, **JADE** implementa en cada plataforma los dos agentes responsables de su administración: los agentes Domain Facilitator y Agent Management System, más bien conocidos por DF y AMS, respectivamente. AMS conoce la dirección de cualquier agente. DF conoce los servicios que proporcionados por cada uno.

JADE está constituido por una serie de clases agrupadas en paquetes. Los principales paquetes de que dispone son:

□ **jade.core**

^{§§} **POO** es el acrónimo de Programación Orientada a Objetos.

*Este paquete constituye el kernel del sistema. Contiene la clase agente, cuya extensión permite construir agentes. También contiene el subpaquete **jade.core.behaviours** contenedor de la jerarquía de **behaviours** o comportamientos. Los comportamientos son la forma natural de expresar que se quiere hacer. Además, permiten por composición de otros comportamientos más simples, poder realizar tareas complejas que incluso pueden ser concurrentes.*

❑ **jade.lang**

*Este paquete contiene un subpaquete para cada lenguaje utilizado en JADE. Además, proporciona una clase para manipular los mensajes **ACL**, la clase **ACLMessage**.*

❑ **jade.content**

Este paquete está constituido por clases auxiliares, cuyo objetivo es proporcionar soporte para la construcción de ontologías y, representación de mensajes mediante expresiones de algún lenguaje soportado por JADE .

❑ **jade.domain**

*Este paquete contiene una serie de clases que representan las entidades administrativas del sistema, esto es, los agentes **DF** y **AMS**.*

❑ **jade.gui**

*Este paquete agrupa un conjunto de clases para construir **GUI's** para los agentes.*

❑ **jade.mtp**

Este paquete proporciona clases para manejar mensajes de agentes externos de otras plataformas.

Además, **JADE** contiene agrupadas en el paquete **jade.tools** un conjunto de herramientas que facilitan el desarrollo de aplicaciones y administración de la plataforma, como por ejemplo:

- ❑ **Remote Management Agent [RMA]**: proporciona una interfaz gráfica para facilitar la administración y control de la plataforma. Es el único agente que no pertenece a la aplicación.
- ❑ **Dummy Agent**: es una herramienta de monitorización y depuración, constituida por una interfaz gráfica y un agente **JADE**. La interfaz gráfica permite crear mensajes **ACL** y

mandarlos a otros agentes. A parte, también, permite disponer de una lista con los mensajes mandados y sus respectivas respuestas, por parte de los agentes receptores.

- ❑ **Introspector Agent:** es un agente que permite monitorizar el ciclo de vida y los mensajes de un agente.
- ❑ **Sniffer:** es un agente capacitado para interceptar mensajes **ACL** al vuelo y visualizarlos gráficamente, usando, una notación muy parecida a los gráficos de **secuencia UML**^{***}. Este agente es muy útil para depurar las conversaciones entre agentes por mostrar como se intercambian los mensajes entre sí.
- ❑ **SocketProxyAgent:** es un simple agente que actúa como pasarela bidireccional entre una plataforma JADE y una conexión **TCP/IP**. Los mensajes ACL, que viajan por la plataforma, son convertidos a cadenas de texto **ASCII** y enviados a través de la conexión vía **socket**. Inversamente, los mensajes ACL pueden ser recibidos a través de una conexión **TCP/IP** y redirigidos a la plataforma. Este agente es útil para manipular **firewalls** o para proporcionar interacciones entre **applets** Java dentro de un navegador.

Finalmente, relataré porque se ha usado la plataforma JADE para desarrollar el SMA propuesto:

- ❑ Es gratuito.
- ❑ Es de código abierto. El código fuente es proporcionado al programador.
- ❑ Posee documentación. En cada versión de JADE son proporcionados un manual del programador y del administrador bastante completos.
- ❑ Sigue las especificaciones de la **FIPA**.
- ❑ Ofrece mejoras y actualizaciones continuas. Cada año surgen nuevas actualizaciones de software. En la actualidad, JADE acaba de desarrollar la versión **2.6**.

4.1 IMPLEMENTACIÓN DE UN AGENTE

^{***} **UML**, acrónimo de Unified Modelling Language, es un conjunto de especificaciones para representar sistemas software orientados a objetos(OO). Para mayor información ver [9].

La clase agente constituye la clase base para definir agentes software. Por tanto, crear agentes dentro de la plataforma JADE consistirá simplemente en extender la clase *agent* y hacer instancias de la nueva clase. Por ejemplo,

```
import jade.core.Agent;  
  
public class myAgent extends Agent {}
```

La clase agente proporciona los métodos necesarios para realizar las operaciones básicas con la plataforma de agentes o AP (tales como registrarse con DF, por ejemplo), y métodos, para añadir comportamientos a un agente. Además, proporciona soporte para el ciclo de vida completo de un agente, incluyendo nacimiento, muerte y suspensión de éste. Los agentes podrán estar en alguno de los siguiente estados:

- ❑ **AP_INITIATED**: El agente aún no se ha registrado con el AMS. No tiene nombre ni dirección asignadas.
- ❑ **AP_ACTIVE**: El agente se ha registrado y está ejecutando sus comportamientos asociados.
- ❑ **AP_SUSPENDED**: El agente está suspendido, ninguno de sus comportamientos es ejecutado.
- ❑ **AP_WAITING**: El agente está bloqueado esperando algún evento, como por ejemplo la llegada de un mensaje.
- ❑ **AP_DELETED**: El agente ha muerto; se ha desregistrado de AMS.

La clase agente dispone de métodos para cambiar el estado de un agente, como por ejemplo:

- ❑ **Los métodos doXXXX**: que fuerzan las transiciones de un estado a otro. Por ejemplo, el método **doWait()** bloquea todos los comportamientos de un agente y hace que pase de “Activo” al estado de “Espera”.

El modelo computacional de una agente es la *multitarea*, donde las tareas son ejecutadas de forma no determinista según una política de planificación. Por tanto, cada servicio que ofrezca un agente deberá ser modelado como un *comportamiento* o tarea. Más adelante, hablaremos con más profundidad sobre que son los comportamientos. De momento, diremos, que un comportamiento es un clase que encapsula una tarea o acción. Todos los comportamientos de un agente, son ejecutados de forma automática por un planificador interno y oculto al programador. La siguiente sección de este capítulo, la **sección 4.2**, será la encargada de aclarar nuestras dudas con respecto a los comportamientos.

Una vez de vuelta al motivo de esta sección, la clase agente proporciona dos métodos públicos básicos:

- **setup()** : Este método permite inicializar un agente. Cuando este método es ejecutado, el agente ya ha sido registrado con el AMS y está casi listo

para iniciar su actividad. Entonces, este procedimiento de inicialización es usado para:

- ❑ describir los servicios proporcionados por el agente y registrarlo con el DF o a otros dominios alternativos.
- ❑ Añadir tareas o comportamientos al agente usando el método `addBehaviour()`. Éstos son ejecutados justo después de que `setup()` alcance su llave de finalización.
- **takeDown()** : Este método puede ser sobrescrito para realizar operaciones de limpieza cuando el agente lo requiera. JADE invoca este método cuando el agente está apunto de ser destruido y sus recursos liberados. De este modo, este método proporciona un mecanismo para desregistrar el agente del DF (el agente es automáticamente desregistrado del AMS) etc.

Los agentes de cualquier aplicación han de ser escritos explotando la capacidad de esta clase.

4.2 COMPORTAMIENTOS

Una vez explicado como se definen los agentes en **JADE**, es necesario comentar como se pueden añadir tareas que les dotan de funcionalidad. Éste es el motivo de discusión de esta sección. Por tanto, en esta sección, hablaremos de la forma en que **JADE** permite a los agentes ejecutar sus tareas. Veamos cómo lo hace.

Un agente debe ser capaz de mantener a la vez varias conversaciones, y, ejecutar varias tareas en respuesta a estos eventos. Por tanto, cada agente está compuesto de varios behaviours o comportamientos, que son ejecutados en un hilo común: el *thread* o hilo del agente.

El programador que quiera dotar de funcionalidad a un agente debe modelar su actividad a través de un o más comportamientos. Estos se implementan como cualquier objeto JAVA. Por tanto, definir comportamientos consiste simplemente en extender alguna subclase de la jerarquía de comportamientos, hacer una instancia de ella y añadir el nuevo comportamiento a la cola de tareas. La clase Agente proporciona dos métodos: **addBehaviour** y **removeBehaviour**, que permiten añadir y eliminar comportamientos a y de un agente específico.

Cada comportamiento definido en un agente tiene asociado un método, denominado **action()**, cuyo cuerpo contiene las instrucciones que serán ejecutadas cuando el comportamiento sea puesto en “**RUN**”.

La política de planificación realizada entre los comportamientos disponibles en la cola de “**READY**” es **round-robin** no apropiativa. Además, cada comportamiento es ejecutado por el planificador hasta finalizar su método **action()**. Esto implica, que el

agente no tenga bucles infinitos, ni en general, operaciones que consuman mucha CPU dentro de **action()**, para permitir que otros comportamientos puedan ejecutarse.

Además, el contexto no es guardado en cada ejecución de los comportamientos, puesto que el método **action()**, es siempre ejecutado por el planificador de principio a fin sin ningún tipo de interrupción. Así pues, los comportamientos son los responsables de mantener su contexto almacenado en variables de instancia.

Puesto que el método **action()** de todos los comportamientos es ejecutado de forma atómica, JADE proporciona otro método denominado **done()**, el cual es invocado justo después de que **action()** finalice. El objetivo de este método es proporcionar un mecanismo para comprobar si un comportamiento ha finalizado su tarea o no. Por tanto, sobrescribiendo este método podemos permitir varias ejecuciones de **action()** y modelar los comportamientos de un agente como una máquina de estados finitos, donde, el estado es mantenido en sus variables de instancia.

Por ejemplo,

```
Public class myStep3Behaviour {
    Private int state = 1;
    Private boolean finished = false;

    Public void action(){
        Switch(state) {
            Case 1: { op1(); state++; break; }
            Case 2: { op2(); state++; break; }
            Case 3: { op3(); state = 1 finished = true;
break; }
        }
    }

    Public boolean done() {
        Returned finished;
    }
}
```

JADE proporciona varias subclases de comportamientos predefinidos, que pueden contener a su vez subcomportamientos. La jerarquía de clases es mostrada en la siguiente figura:

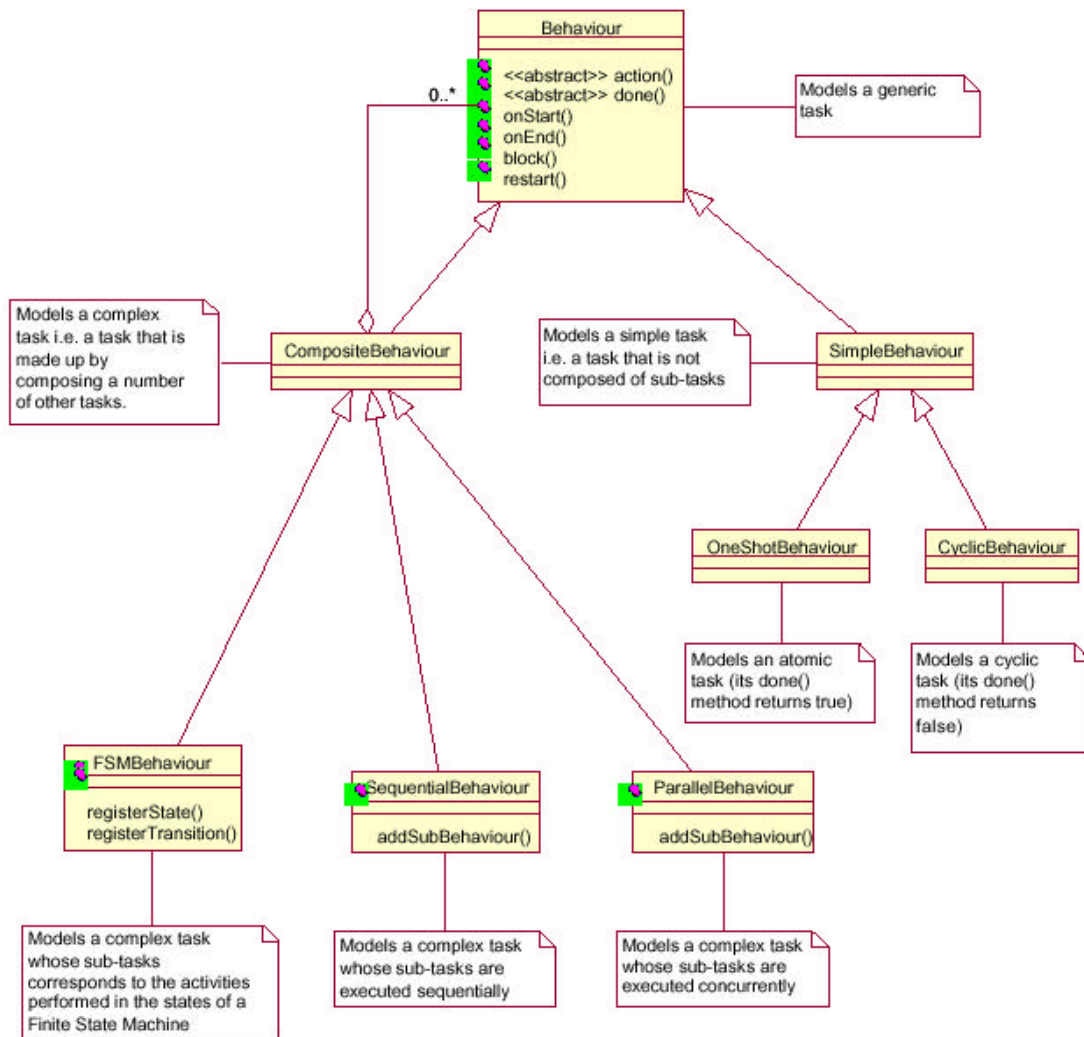


Figura 8. Jerarquía de Comportamientos

❑ clase **SimpleBehaviour**

Esta clase abstracta puede ser sobrescrita por una subclase para modelar un comportamiento simple y ejecutado sin interrupción. Contiene dos subclases:

o clase **OneShotBehaviour**

Esta clase abstracta modela un comportamiento atómico que es ejecutado una sola vez por el planificador (habitualmente, cuando un evento ocurre). Su método *done()* siempre devuelve cierto.

o clase **CyclicBehaviour**

Esta clase abstracta modela comportamientos que son ejecutados por el planificador de manera cíclica. Su método *done()* siempre devuelve falso.

❑ clase **CompositeBehaviour**

Esta clase abstracta es usada para modelar comportamientos contruidos por composición de otros comportamientos denominados subcomportamientos. De este modo, las operaciones que serán realizadas no son definidas en el propio comportamiento, pero sí, dentro de sus subcomportamientos. El comportamiento compuesto sólo se preocupará de la política de planificación de sus subcomportamientos. Esta clase se puede dividir en tres subclases:

○ clase **SequentialBehaviour**

Esta subclase de *CompositeBehaviour* es una clase abstracta que ejecuta sus comportamientos hijo de forma secuencial. Esto es, uno después de otro.

○ clase **PararellBehaviour**

La principal característica de esta subclase, como su nombre indica, es que ejecuta sus subcomportamientos de manera concurrente finalizando cuando una condición final es alcanzada. Esta clase es indicada cuando las tareas pueden ser divididas en un conjunto de operaciones alternativas que buscan alcanzar un objetivo común.

○ clase **FSMBehaviour**

Esta clase ejecuta sus comportamientos asociados como una máquina de estados finitos. Por tanto, esta clase permite definir cuáles son las transiciones entre estados e indicar cuáles son los estados finales. Cada estado es modelado por un subcomportamiento.

❑ clase **SenderBehaviour**

Esta clase extiende a la clase *OneShotBehaviour* y es usada para enviar un mensaje. El mensaje ACL [**Agent Communication Language**] debe ser pasado como parámetro en su constructor.

❑ clase **ReceiverBehaviour**

El propósito principal de este comportamiento es encapsular la acción de recibir mensajes. Su actividad finaliza cuando el mensaje adecuado es recibido. Este comportamiento se bloquea hasta que un nuevo mensaje está disponible, pero, sin paralizar la actividad del agente. Además, permite especificar (con los operandos lógicos y / o / no) condiciones sobre los mensajes a recibir.

□ clase **WakerBehaviour**

Esta clase abstracta implementa una tarea que es ejecutada una sola vez cuando el *timeout* es excedido.

Notar que los comportamientos y subcomportamientos pueden ser añadido (usando *addBehaviour()*) cuando sea necesario e incluso de forma dinámica.

4.3 PROTOCOLOS DE INTERACCIÓN

En secciones anteriores, hemos discutidos sobre qué es un agente. También, hemos explicado como se les puede añadir funcionalidad mediante comportamientos. Por tanto, sólo nos falta hablar de cómo **JADE** permite manipular los protocolos de comunicación. Veamos cómo lo hace:

JADE implementa casi todos los protocolos estándares definidos por la FIPA. Esto permite construir todo tipo de conversaciones entre agentes. Así, JADE distingue el papel del *Initiator* (el agente que inicia la conversación) del papel del *Responder* (el agente que se une a la conversación tras entrar en contacto con el primero), de forma similar a las especificaciones FIPA. Pero además, proporciona comportamientos para ambos papeles. Es decir, proporciona comportamientos para manejar los estados del agente iniciador [**Initiator Behaviours**] por una banda, y, comportamientos para manejar los estados correspondientes al agente participante [**Responder Behaviours**] por la otra.

Los Initiator Behaviours se caracterizan por ser eliminados de la cola de tareas tan pronto como alcanzan el estado final del protocolo de interacción. Sin embargo, los Responder Behaviours se caracterizan por comportarse de forma cíclica, es decir, por ser relanzados continuamente por el planificador.

JADE implementa los protocolos de interacción FIPA-request, FIPA-query, FIPA-propose, FIPA-request-when, FIPA-recruiting, FIPA.-brokering, FIPA-subscribe, etc. con la ayuda de un par de clases: **AchieveREInitiator/Responder**. Esto es posible porque todos ellos comparten la misma estructura. En la siguiente figura, podemos observar la estructura homogénea de estos protocolos.

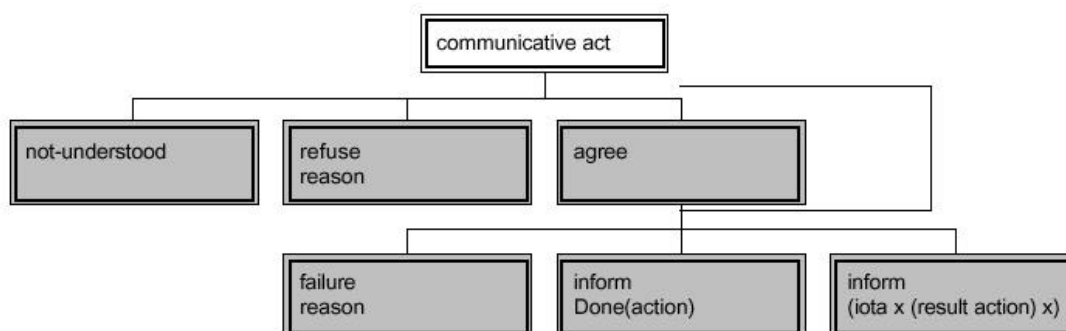


Figura 9. Estructura de los protocolos de interacción

JADE permite hacer opcional la transmisión de las confirmaciones positivas. De esta forma, la productividad del sistema aumenta si las acciones son cortas (mandar un **agree** supone desperdiciar mucho tiempo en vano e incrementar el **overhead**^{†††}).

Ambas clases contienen métodos de *callback*^{‡‡‡} para manipular los estados del protocolo, y en particular, para preparar respuestas en el caso del agente participante.

La implementación de estas clases soporta *timeouts* sobre las primeras respuestas, tales como un mensaje **refuse**. Sin embargo, si las aplicaciones necesitan fijar tiempos límite o *deadlines* para recibir los últimos mensajes, estos habrán de ser expresados dentro del contenido del mensaje (definiendo los términos dentro de una *ontología*).

El protocolo FIPA Contract Net, el cual permite al iniciador enviar un mensaje **Call For Proposal** a un conjunto de agentes, evaluar sus propuestas y aceptar la mejor o incluso rechazarlas todas, es implementado vía la clase **ContractNetInitiator/Responder**. Esta clase funciona de forma similar a las anteriores.

^{†††} **Overhead:** se utiliza para medir el tanto por ciento de tiempo utilizado por el sistema para llevar a cabo una tarea. Es decir, es el tiempo que el sistema necesita para gestionar una acción y en el cual no se puede ejecutar el código útil de ésta. Se define como tiempo total empleado en realizar la tarea menos el tiempo real de ejecución de dicha tarea.

^{‡‡‡} **Callback:** por métodos *callback*, se entienden todos aquellos métodos escritos por el programador que son llamados por la aplicación en respuesta a algún evento.

5. Justificación de las técnicas utilizadas

La siguiente sección pretende explicar porque se han escogido las anteriores tecnologías para el desarrollo de este proyecto. Esta sección estará dividida en cuatro partes. En la primera sección describiremos porque se ha escogido *RDF* como lenguaje de representación de ontologías. En la siguiente sección, debatiremos porque se ha usado *RDF* en detrimento de *XML*. A continuación, explicaremos porqué se han usado agentes. Finalmente, en la última sección, redactaremos las conclusiones del uso de ambas tecnologías.

5.1 Porqué RDF?

En la actualidad, les diferentes propuestas de metadatos pueden ser vistas como una solución a uno de los conflictos más duros de la **World Wide Web** existente hoy en día: *la gran dificultad que existe al intentar automatizar tareas que realizan trabajo útil en la web*. Sin ir más lejos, la web desde sus orígenes fue diseñada para el consumo humano de información, tal como demuestra el simple hecho de que la única forma de búsqueda en la web se restringe al *matching* de cadenas de texto o palabras clave. Cualquier persona que haya utilizado un buscador, se habrá percatado que introducir unas cuantas palabras clave y recibir multitud de aciertos no siempre es cosa útil. Casi siempre es necesario realizar un paso previo de eliminación de redundancias o filtraje de documentos; incluso podría ocurrir que dichas palabras claves no fueren prominentes en los documentos en sí relevantes.

Una solución plausible al problema de la búsqueda de información – y en general, sobre permitir agentes vagar por la web realizando trabajo útil – sería proporcionar un mecanismo que permitiera precisar descripciones sobre los documentos disponibles en la web. Esto permitiría convertir la web, en general, de un simple medio de consumo de información, en un medio dónde las máquinas serían capaces de interpretar la información circulante. Este mecanismo podría denominarse **RDF – The Resource Description Framework** [Lassila & Swick 1997] –.

RDF se caracterizó desde sus inicios por enfatizar el procesado automático de información en la web. Esto es, permitir que aplicaciones autónomas sean capaces de interpretar información en la web sin la continua intervención de agentes humanos, y en definitiva, los únicos capaces de poder establecer una relación mental entre los términos y aquello que representan. Por ejemplo, **RDF** podría proporcionar mejores capacidades de búsqueda a través del diseño de ontologías y la descripción **RDF** de los recursos de la web. También, podría emplearse en la catalogación de recursos web, proporcionando ontologías que describieran el contenido y las relaciones de contenido disponibles. Como vimos anteriormente, **RDF** dispone de un mecanismo extensible de tipos capaz de expresar las relaciones existentes entre diferentes recursos como si de un lenguaje orientado a objetos se tratara. También, se podría utilizar en las calificaciones del

contenido de simples páginas e incluso en la descripción de colecciones de páginas que representan un “documento” lógico individual. Y sobretodo, por agentes inteligentes software para facilitar el intercambio y compartido de conocimiento.

Sin embargo, la elección de RDF se ha fundamentado en su capacidad para proporcionar un mecanismo capaz de crear vocabularios y ontologías, en definitiva, la moneda de intercambio de nuestros agentes dentro del SMA. Debido a su expresividad y capacidad para construir jerarquías de clases, RDF ha permitido representar los términos de búsqueda por un lado y, la descripción, incluyendo calificación, de los recursos encontrados en la red por el otro. Es decir, se ha podido construir un buscador cuya capacidad de búsqueda se ha incrementado al poder definir clases y propiedades (al mejorar el conjunto de términos de búsqueda se mejora la capacidad de selección) y que además, es capaz de puntuar y describir los recursos encontrados según una ontología predefinida. Como podéis comprobar, propiedades que se corresponden a las áreas de descubrimientos de recursos y de catalogación, mencionadas anteriormente.

En particular, y con respecto a las arquitecturas de agentes que trabajan en la web, la habilidad de describir objetos para propósitos de interoperabilidad, constituye el corazón del asunto.

5.2 Porqué RDF y no XML?

El uso de RDF en este proyecto en lugar de otras tecnologías como XML, viene determinada porque XML permite únicamente extraer datos, mientras que RDF permite extraer el contenido y además, en cierta manera, el significado.

Supongamos, el siguiente ejemplo sencillo:

Queremos representar la sentencia *"El autor de la página es Ora"*.

En el modelo de datos RDF tradicional esto se representaría con la **terna(autor, página, Ora)**. Hecho que permitiría almacenar la sentencia en forma de sujeto, predicado y objeto y mantener así su semántica; Que a su vez, permitiría formular preguntas sobre el modelo, como “¿Es Ora el autor de la página?”, sin necesidad de conocer la estructura de ningún documento.

Sin embargo, en XML esta información típicamente como se representaría?

```
<autor>
  < uri>página< /uri>
  <nombre>Ora< /nombre>
< /autor>
```

o quizás

```

<documento href="página">
    <autor>Ora</autor>
</documento>

```

o quizás

```

<documento>
    <detalles>
        <uri>href="página"</uri>
        <autor>
            <nombre>Ora</nombre>
        </autor>
    </detalles>
</documento>

```

o quizás

```

<documento>
    <autor>
        <uri>href="página"</uri>
        <detalles>
            <nombre>Ora</nombre>
        </detalles>
    </autor>
</documento>

```

```

<documento href="http://www.w3.org/test/page" autor="Ora" />

```

Todos son documentos XML correctos y para una persona que leyese la información tendrían todos el mismo significado. Sin embargo, para un ordenador que leyese esta información, todos estos árboles XML serían distintos.

Supongamos un árbol XML como sigue:

```

<v>
    <x>
        <y> a="ppppp"</y>
        <z>
            <w>qqqqq</w>
        </z>
    </x>
</v>

```

Sin mirar el esquema, sólo conocemos la estructura del documento, nada más. No podremos deducir nada, ni tan siquiera las relaciones que hay entre los diferentes elementos del documento. Esto es, seremos incapaces de decir si ppppp es una y de qqqqq, por ejemplo. Incluso no sabremos qué preguntas reales hacer y nos limitaremos a hacer preguntas del estilo: ¿Cuál es el contenido de w dentro de z? o ¿Cuál es el contenido de w dentro del primer elemento de y que tiene como atributo ppppp?. En definitiva, preguntas sobre la estructura del documento. Así pues, tener que hacer una pregunta tan simple como si *“Ora es el autor de la página”* hará que se puede formular

de tantas formas cómo árboles XML posibles. Por tanto, preguntar en XML, consistirá en resumir todas las posibles representaciones en una de sola, que de hecho, es lo que hace RDF. RDF simplemente proporciona varias formas estándar de escritura de sentencias, pero todas, en última instancia, provocan el mismo efecto en términos RDF. Es decir, un único árbol RDF, el cual, puede provenir de varios árboles XML, pero no al revés.

RDF, en general, es mucho más flexible, y de hecho, distintas representaciones admiten el mismo conjunto de preguntas sin ningún inconveniente. Incluso si en un futuro, se definiesen otras formas estándar de escritura, el mismo conjunto de preguntas continuaría siendo válido. Para entender esto, cojamos el ejemplo anterior:

la

terna(autor, página, Ora)

e intentemos representarla usando sintaxis RDF. Esta representación podría dar lugar a:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/TR/WD-rdf-syntax#"
  xmlns:s="http://docs.r.us.com/bibliography-info#" / >
  <rdf:Description about="http://www.w3.org/test/page"
    s:Author="http://www.w3.org/staff/Ora" / >

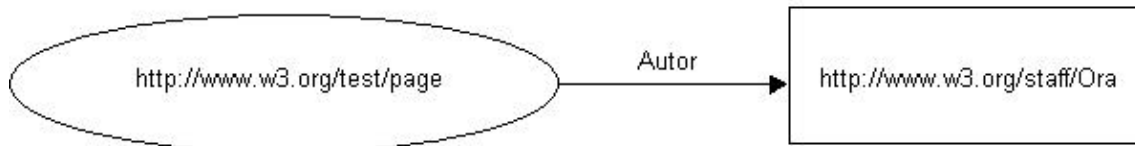
</rdf>
```

o quizás a

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/TR/WD-rdf-syntax#"
  xmlns:s="http://docs.r.us.com/bibliography-info#" / >
  <rdf:Description about="http://www.w3.org/test/page">
    <s:Author rdf:resource="http://www.w3.org/staff/Ora" / >
  </rdf:Description>
</rdf>
```

perteneciente la primera a la sintaxis abreviada y la segunda a la sintaxis RDF / XML serializada habitual. Sin embargo, a pesar de ser distintas, las dos representaciones anteriores convergen en la misma terna: la **terna(autor, página, Ora)**.

Y por tanto, gráficamente, se pueden ilustrar con un sólo grafo:



Así pues, resumiendo, podemos decir que RDF funciona como un modelo semántico de datos capaz de permitir preguntas referentes a su contenido y no a la estructura del documento. Esto permite en contraposición con XML:

- ❑ Parsear el árbol semántico constituido por temas, recobrando así, la información d'aquellas que conocemos e ignorar aquellas que no entendemos.

Además, XML presenta una serie de problemas por basar su entendimiento en la estructura de los documentos. Estos son:

- ❑ Sin tener en cuenta el problema de obtener un esquema, o tener una aplicación programada para reconocer un tipo particular de documento, XML no permite extraer información semántica de un documento.
- ❑ Cuando un esquema XML cambia, por ejemplo, introduce nuevos elementos intermedios, puede invalidar alguna consulta anterior que haya sido basada en la estructura del documento.

Además, también falla en un objetivo bastante importante, no mencionado anteriormente, *la escalabilidad*:

- ❑ El orden en el cual los elementos aparecen en el documento XML es significativo y muchas veces necesario. Esto es altamente antinatural en el mundo de los metadatos. Nadie se preocupa si el director o el título de una película es listado primero siempre y cuando ambos datos estén disponibles para realizar búsquedas.
- ❑ Además, mantener el orden correcto de los elementos de datos es caro y difícil.

De este modo, para nuestra aplicación donde es imprescindible el uso e intercambio de ontologías entre agentes, se ha escogido RDF en contraposición con XML por los siguientes motivos, justificados, con anterioridad:

- ❑ **Flexibilidad:** esta característica es fundamental. Puesto que nuestro buscador se fundamenta en el uso de ontologías para mejorar el

motor de búsqueda y el mecanismo de composición de resultados, es necesario, disponer de un mecanismo que permita añadir, modificar o eliminar vocabularios sin tener que realizar grandes modificaciones. En este sentido, RDF proporciona la flexibilidad que XML no dispone (añadir un elemento intermedio en un vocabulario significa generar código nuevo, mientras en RDF, se podría continuar obteniendo el mismo resultado).

- ❑ **Eficiencia:** RDF permite preguntar al modelo de forma más eficiente que XML. Basta con conocer algún elemento (predicado, sujeto o objeto) para seleccionar una terna de forma rápida. En XML, habríamos de parsear la información siguiendo la estructura del documento conforme a un esquema(DTD).
- ❑ **Escalabilidad:** relacionada con la flexibilidad. Si en un futuro, nuestro buscador estuviera destinado a interpretar metadatos en la web, usar XML comportaría buscar manteniendo el orden de los elementos. Esto, provocaría llegar a convenios con las diversas organizaciones para preservar ese orden.

5.3 Porqué Agentes?

Las tecnologías de agentes y en especial los sistemas multiagentes (dónde una comunidad de agentes interaccionan unos con otros) ofrecen mayores promesas de flexibilidad y capacidad que las aplicaciones distribuidas tradicionales. Por tanto, y, teniendo en cuenta su habilidad para interaccionar con sistemas remotos de todo tipo y, realizar tareas de forma autónoma sin la constante comunicación con el usuario, podemos decir que los agentes cumplen la condición fundamental de un sistema con capacidad para procesar el masivo volumen de información disponible en el **World Wide Web**. Además, los agentes proporcionan mayor flexibilidad que las aplicaciones existentes hoy en día, puesto que pueden ser programadas de manera inteligente para reaccionar delante situaciones y entornos desconocidos en tiempo de diseño. Por tanto, en un entorno tan variable como el mundo de Internet, las aplicaciones diseñadas con agentes serán capaces de adaptarse a los cambios rápidamente. Y además, ante las nuevas situaciones serán fácilmente incorporables nuevos servicios que las manejen. Por ejemplo, los agentes de Internet son capaces de manejar **HTTP**^{§§§}, pero en un futuro podrían manejar servicios **FTP** o procesadores de XML, por ejemplo.

Otras ventajas del diseño con SMA:

- ❑ Las funciones o tareas pueden ser repartidas entre los diferentes agentes según su complejidad; constituyen sistemas modulares.

^{§§§} **HTTP**: acrónimo de HyperText Transfer Protocol, es el protocolo estándar para el intercambio de hipertexto entre máquinas remotas vía Internet.

- ❑ Presentan un mayor grado de fiabilidad por su naturaleza distribuida y en consecuencia son más tolerantes ante fallos que cualquier sistema centralizado.
- ❑ Son más eficientes porque las tareas pueden realizarse de forma simultánea por varios agentes y por tanto, mejoran el rendimiento. Esta propiedad es fundamental en aplicaciones, como nuestro buscador, que necesitan rendimiento.
- ❑ Son sistemas más optimizados, puesto que los agentes se pueden añadir y eliminar según las necesidades y los recursos disponibles e incluso de forma automática a partir del conocimiento del estado del sistema por los propios agentes.

5.4 Conclusiones

El desarrollo de cualquier aplicación que de forma automática recobre información del **World Wide Web**, requiere que se superen dos deficiencias importantes:

- ❑ La primera es poder disponer de un mecanismo de descripción de recursos que permita expresar conocimiento (y por tanto, supere la barrera impuesta desde la creación de Internet: *la asunción de ser un medio para humanos*).
- ❑ La segunda, en cambio, hace referencia a poder disponer de una infraestructura autónoma que sea capaz de reaccionar y adaptarse a los rápidos cambios de la web, como por ejemplo, cambios en su topología, y además, proporcione un buen nivel de rendimiento.

Como podemos deducir de los párrafos anteriores, ambas deficiencias se pueden salvar empleando **RDF** y **SMA**, respectivamente. Por tanto, para elaborar nuestro buscador, necesitaremos combinar el poder de descripción de **RDF** y la autonomía y eficiencia de los **SMA**.

6. Implementación del SMA

Esta sección pretende explicar como se han implementado los agentes involucrados en el sistema, sus tareas y los protocolos de comunicación usados entre ellos.

Recordemos, que los agentes a implementar eran:

- Agente de Internet($A_{internet}$)
- Agente Coordinador(A_{coord})

Llegados a este punto, el paso más lógico sería hacer una descripción de cada agente de forma individual, explicando como se han solucionado los diferentes problemas. Sin embargo, puesto que ambos agentes han de tratar con ontologías **RDF**, en las siguiente líneas haremos una breve descripción del procesador **RDF** seleccionado. Más adelante, ya nos ocuparemos de la descripción de los agentes. Entonces, para empezar, diremos que **JENA** ha sido la herramienta seleccionada.

6.1 JENA API

JENA es una *API* Java desarrollada para procesar **RDF**. Este API está definida en términos de *interfaces*, para permitir que las aplicaciones pueden trabajar con los diferentes implementaciones sin cambio. Los paquetes contienen *interfaces* para representar recursos, propiedades y cualquier elemento **RDF**. Por tanto, **JENA**, de forma análoga a **JADE**, no es más que un conjunto de paquetes para procesar **RDF**. Los paquetes son los encargados de salvar las dificultades de bajo nivel, pero es responsabilidad de *programador* dotar a las aplicaciones de funcionalidad..

JENA soporta a través de sus paquetes gran variedad de operaciones. Estas van desde *leer* documentos **RDF** y *almacenarlos* en forma de ternas (**grafo**), hasta operaciones binarias tales como *uniones*, *intersecciones* y *diferencias* entre grafos. Es por tanto, una **API** muy completa.

Sin embargo, la capacidad más sorprendente de **JENA**, es la posibilidad de poder realizar consultas sobre un grafo (*almacenado en forma de ternas*) usando un lenguaje parecido a **SQL**. Este lenguaje, denominado **RDQL**, es una implementación de **SquishQL RDF query language**. Si **RDF** proporciona grafos dirigidos, **RDQL** proporciona una forma de especificar patrones que contrastados con los grafos, permiten

obtener resultados. De forma similar a los cursores de *JDBC*****, los resultados son devueltos en una instancia de *ResultBinding*. Este objeto liga las variables de la consulta con los resultados.

6.1.1 Sintaxis RDQL

En *SQL*, una base de datos es un mundo cerrado. La cláusula *FROM* identifica las tablas en la base de datos. La cláusula *WHERE* identifica las restricciones de la consulta que pueden ser extendidas con *AND*. De forma similar, en *RDQL*, la web constituye la base de datos y la cláusula *FROM* identifica el grafo *RDF*. Las variables se definen precedidas con ‘?’ y las URIs son representadas entre <>. Las consultas se definen usando las siguientes palabras clave:

- ❑ **SELECT**: identifica las variables que serán devueltas a la aplicación. Si no todas las variables son requeridas, entonces, especificar las necesarias puede reducir bastante la cantidad de memoria necesaria para obtener resultados.
- ❑ **FROM**: esta cláusula especifica sobre que grafo se hace la consulta. El valor ha de ser una URI.
- ❑ **WHERE**: especifica el patrón mediante una lista de ternas.
- ❑ **AND**: especifica expresiones booleanas.
- ❑ **USING**: Es una forma de acortar la longitud de las URIs. Es un simple mecanismo de abreviación para URIs largas a través de prefijos.

Para entender estos conceptos veamos un ejemplo:

```
SELECT ?resource , ?familyName
FROM file: rdf/family.rdf
WHERE  (?resource, <info:age>, ?age) ,
        (?resource, <vCard:N>, ?y) , (?y, <vCard:Family, ?familyName)
AND ?age >= 24
USING  info    FOR <http://somewhere/peopleInfo> ,
        Vcard  FOR <http://www.w3.org/2001/vcard-rdf/3.0#>
```

Este ejemplo, devuelve el URI y el nombre de familia de todos aquellos recursos que se encuentran en el fichero “rdf/family.rdf”, tienen una propiedad *edad* y una propiedad *nombre de familia* y además, son mayores de 24 años. La cláusula *USING* sirve como método de abreviación de las URIs largas. Si esta cláusula no se hubiera utilizado, en lugar de <info:age> se habría de escribir el URI completo: <http://somewhere/peopleInfo#age>.

**** *JDBC* es una API de Java para manipular Bases de Datos. Permite realizar operaciones comunes de forma simple, tales como, sentencias *SELECT*.

Una vez explicados estos conceptos previos, ya podemos pasar a comentar los agentes de forma individual. En este momento, el lector, habría de ser capaz de entender las consultas **RDQL** introducidas para procesar las ontologías.

6.2 Agente de Internet($A_{internet}$)

En esta sección explicaremos como se ha definido el agente de Internet en vistas a realizar las tareas asignadas. Recordemos que la principal tarea de este agente es recobrar información de la web y elaborar una ontología con los resultados. Veamos como la realiza.

El agente de Internet realiza las siguientes tareas en su método *setup*:

- ❑ Registra sus servicios al DF de la plataforma: “InternetAgent”.
- ❑ Inicializa su interfaz gráfico.
- ❑ Instancia un objeto de la clase *Ontology*. Este objeto contiene las ontologías de búsqueda almacenadas.
- ❑ Inicializa el objeto lector de **RDF** mediante una instancia de la clase *RDFReader*.
- ❑ Añade una instancia del comportamiento *ServerBehaviour*.

El motivo de registrar el agente al DF es para indicar al agente coordinador que se ha añadido un nuevo agente de Internet que puede usar. De esta forma, el agente A_{coord} o agente coordinador, consultando al agente DF, será capaz de asignar peticiones de búsqueda sobre los agentes de Internet.

El comportamiento *ServerBehaviour* es el responsable de que el agente de Internet funcione como un *servidor*. Este comportamiento se ejecuta de forma cíclica. Es decir, el valor de *done()* es siempre *falso*. El funcionamiento es el siguiente:

Al iniciarse *action()*, el comportamiento se bloquea mediante *blockingReceive()* aguardando la llegada de una mensaje tipo FIPA-Request. Esta llamada bloquea toda la actividad del agente, pero en este caso, esto interesa porque el agente no consumirá CPU cuando este ocioso. Cuando el agente recibe un mensaje de tipo Request, empieza actuar. Primero, comprueba si el contenido del mensaje es una ontología RDF válida. Luego, en caso afirmativo, extrae la información y actúa en consecuencia. Finalmente, cuando el comportamiento sale de *action()*, el planificador interno lo vuelve añadir a la cola de tareas al comprobar que es cíclico. De esta forma, el comportamiento funciona como un servidor: escucha una petición, la trata y continúa escuchando.

Este agente puede recibir tres tipos de mensajes en *ServerBehaviour*. Cada uno de ellos se corresponde con uno de los tres tipos de acciones que puede llevar a cabo un $A_{internet}$. Estas acciones son:

- ❑ *Extraer* páginas del **World Wide Web** a partir de una ontología de búsqueda.

- ❑ *Añadir* un conjunto de propiedades a una ontología que ya está en proceso de búsqueda.
- ❑ *Eliminar* una lista de propiedades de una ontología que se está usando, actualmente, para extraer recursos de la red.

Sin embargo, el contenido de estos mensajes sólo puede ser o bien, una ontología de búsqueda especificada por una clase **RDF** con propiedades [**query ontology**] o bien, propiedades aplicadas a una *clase* en proceso de búsqueda. Entonces, es evidente que dos tipos de mensajes comparten el mismo tipo de contenido. Estos son los mensajes que se corresponden con las acciones de *añadir* y *eliminar* propiedades de forma *on-line*.

A continuación exponemos un ejemplo de cada uno de los tipos de mensajes que puede recibir este agente:

- ❑ Mensaje de extracción de recursos a partir de una **query ontology**:

<rdf:RDF

```
xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
xmlns:NS0='http://www.msg.com/message-ont#'
xmlns:rdfs='http://www.w3.org/2000/01/rdf-schema#'
>
```

```
<rdf:Description rdf:about='registeredTo'>
  <rdf:type rdf:resource='http://www.w3.org/1999/02/22-rdf-syntax-ns#Property'/>
  <rdfs:domain rdf:resource='MotorVehicle'/>
</rdf:Description>
```

```
<rdf:Description rdf:about='MotorVehicle'>
  <rdf:type rdf:resource='http://www.w3.org/2000/01/rdf-schema#Class'/>
  <rdfs:subClassOf rdf:resource='http://www.w3.org/2000/01/rdf-
schema#Resource'/>
</rdf:Description>
```

```
<rdf:Description rdf:about='http://www.msg.com/message-ont#ACLMessage'>
  <NS0:action>Search</NS0:action>
  <NS0:threshold>0.5</NS0:threshold>
  <NS0:deadline>60000</NS0:deadline>
</rdf:Description>
```

</rdf:RDF>

- ❑ Mensaje de añadir propiedades a una ontología:

<rdf:RDF

```
xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
xmlns:NS0='http://www.msg.com/message-ont#'
xmlns:rdfs='http://www.w3.org/2000/01/rdf-schema#'
```

>

```
<rdf:Description rdf:about='height'>
  <rdf:type rdf:resource='http://www.w3.org/1999/02/22-rdf-syntax-ns#Property'/>
  <rdfs:domain rdf:resource='#Truck'/>
</rdf:Description>
```

```
<rdf:Description rdf:about='http://www.msg.com/message-ont#ACLMessage'>
  <NS0:action>Add</NS0:action>
</rdf:Description>
```

</rdf:RDF>

- Mensaje de eliminar propiedades a una ontología:

<rdf:RDF

```
  xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:NS0='http://www.msg.com/message-ont#'
  xmlns:rdfs='http://www.w3.org/2000/01/rdf-schema#'
```

>

```
<rdf:Description rdf:about='height'>
  <rdf:type rdf:resource='http://www.w3.org/1999/02/22-rdf-syntax-ns#Property'/>
  <rdfs:domain rdf:resource='#Truck'/>
</rdf:Description>
```

```
<rdf:Description rdf:about='http://www.msg.com/message-ont#ACLMessage'>
  <NS0:action>Remove</NS0:action>
</rdf:Description>
```

</rdf:RDF>

Como se observa en los anteriores mensajes, se ha añadido la descripción de un recurso adicional en cada uno de ellos. Este recurso se denomina **ACLMessage**. El propósito de la adición de este recurso es proporcionar un mecanismo de identificación de cada tipo de mensaje. A través del valor de la propiedad **action**, el agente de Internet podrá determinar que acción debe iniciar. El valor de **action** puede ser una de las siguientes cadenas de texto: *Search*, *Add* y *Remove*. La cadena de texto *Search* determina que el agente debe buscar en Internet a partir de la ontología suministrada. La cadena *Add* indica que el agente debe añadir una lista de propiedades y *Remove* que debe eliminarlas. Además, y en el caso de mensajes de tipo *Search*, aparecen dos propiedades extras. Éstas son **threshold** y **deadline**. La primera, indica el umbral de filtrado, la segunda, el tiempo máximo de búsqueda en milisegundos. Estos son los dos parámetros configurables de cualquier búsqueda.

A continuación expondré, a modo de ejemplo, las sentencias en **RDQL** que permiten extraer la clase y las propiedades de la **query ontology** (Mensaje tipo *Search*). Las demás sentencias, son bastante sencillas y no serán expuestas en esta documentación.

La sentencia que permite extraer el nombre de la clase es:

SELECT ?x WHERE (?x, <RDF.type>, <RDFS.Class>) ,

Esta sentencia devuelve todos aquellos recursos que sean de tipo *clase*. La constantes *RDF.type* y *RDFS.Class* equivalen a los elementos RDF estándares *rdf:type* y *rdfs:Class*, respectivamente. Sin embargo, para nuestro propósito, esta sola sentencia no es suficiente. Como es susceptible de devolver más de una ocurrencia de tipo *clase* o ninguna, se ha tenido que realizar aparte un control sobre la *cardinalidad* del resultado. Una **query ontology** sólo puede tener una clase, por tanto, cualquier otra posibilidad ha sido desechada mediante excepción.

La sentencia que permite extraer las propiedades es la siguiente:

**SELECT ?x WHERE (?x, <RDF.type> , <RDF.Property>),
(?x, <RDFS.domain> , <clase >)**

Esta sentencia devuelve todos aquellos recursos que sean de tipo *propiedad* y que sean aplicables a la *clase* obtenida en la primera sentencia. La constantes *RDF.type*, *RDF.Property* y *RDFS.domain* equivalen a los elementos RDF estándares *rdf:type*, *rdf:Property* y *rdfs:domain*, respectivamente. Aunque la segunda tema de la cláusula *WHERE* no es *a priori* necesaria (A_{coord} sólo coloca propiedades pertenecientes a la clase <clase> en la **query ontology**), su inclusión ha venido motivada por razones *preventivas*. Si una propiedad no pertenece a la clase será ignorada. De esta forma, aumentamos la *fiabilidad* al aumentar la restricción sobre las condiciones de selección.

El procesamiento del contenido *RDF* de los mensajes anteriores es llevado a cabo por la clase *RDFReader*. Esta clase realiza de forma automática el tratamiento de los contenidos, es decir, obtiene los parámetros de entrada de *ACLMessage* y coloca la *ontología de búsqueda* dentro del objeto *Ontology* cuando es necesario.

El objeto *Ontology* es usado como receptáculo de las diversas **query ontology**. Como nuestro agente funciona como un servidor web y además, a de permitir añadir / eliminar propiedades cuando las ontologías están en proceso de búsqueda, es necesario disponer de un objeto que mantenga la información actualizada de éstas. Este objeto es *Ontology*.

Una vez que el comportamiento *ServerBehaviour* ha identificado el tipo de mensaje recibido, empieza actuar. En el caso de *añadir / eliminar* propiedades la operación es trivial. Simplemente, se añaden o se eliminan de *Ontology*. Sin embargo, en el caso de buscar *recursos* en la red, la operación se complica. Para facilitar la explicación dividiremos el contenido en dos partes: en el proceso de búsqueda de links iniciales y en el proceso de extracción. Estas partes se corresponden con los procesos P_{get} y $P_{analysis}$, respectivamente.

6.2.1 El proceso P_{get}

El proceso P_{get} es el proceso responsable de obtener las primeras URL para que el proceso $P_{analysis}$ pueda empezar la extracción. Este proceso se puede realizar de dos formas: una mediante la GOOGLE API y la otra, mediante el análisis de la página HTML generada por el *cgi* de cualquier otro servidor. En primera instancia, hablaremos de la GOOGLE API, después, pasaremos a comentar la segunda opción.

La GOOGLE API es una API proporcionada por GOOGLE para permitir que las aplicaciones pueden obtener las URLs de recursos web de forma directa. Es decir, sin necesitar analizar la respuesta HTML generada por su *cgi*. Esta API se caracteriza porque actúa de manera análoga a como un usuario se conectaría a GOOGLE para buscar información. En primera instancia, enviaría la petición de búsqueda, compuesta por la cadena de texto y un conjunto de parámetros al servidor de GOOGLE Web API service. Después, esperaría los resultados de la búsqueda. Sin embargo, hay una pequeña diferencia. Para poder usar los servicios de esta API, es necesario primero autentificarnos mediante un clave que se obtiene al registrarse en GOOGLE. Una vez obtenida, ya podemos empezar a utilizarla sin ningún tipo de problema.

La segunda opción es menos elegante que la anterior, pero, aporta mayor flexibilidad. No se liga con ningún buscador en concreto. Su funcionamiento consiste en parsear la respuesta HTML del buscador en busca de URLs. Debido a que la respuesta HTML no es homogénea entre buscadores ni incluso dentro del mismo buscador (la presentación varía con el tiempo), se ha desarrollado un mecanismo de análisis por **patrón**. Es decir, el usuario, mediante la herramienta *Setup*, puede diseñar un patrón que será contrastado con la respuesta HTML. Entonces, cada vez que se de una ocurrencia del patrón dentro de la página se obtendrá una URL. De esta forma, cuando cambie la respuesta HTML no será necesario diseñar un nuevo *parser* específico, bastará con cambiar el patrón. Esta segunda opción requiere que se especifiquen dos parámetros:

- ❑ **URL del *cgi***: es necesario especificar cuál es el *cgi* del **buscador** al que hay que conectarse. Sin la URL del *cgi* no podremos enviarle la cadena de texto con la petición de búsqueda, y por tanto, no nos generará ninguna página de respuesta.
- ❑ **Etiqueta <A> del patrón que contiene la URL válida**: como que un *patrón* puede contener más de una etiqueta <A> , es necesario especificar cuál de ellas contiene la URL válida. Es obligatorio siempre incluir al menos una etiqueta <A> dentro del patrón..

Una vez comentadas las dos opciones, sólo falta decir qué cadena de búsqueda se utiliza en las peticiones. La respuesta es que se utiliza simplemente el nombre de la **clase** de la **query ontology**.

6.2.2 El proceso $P_{analysis}$

El proceso $P_{analysis}$, responsable de extraer toda la información posible de los websites, ha sido diseñado teniendo en cuenta el viejo dicho de *divide y vencerás*. Pero en este caso, con otro sentido:

La precondition de este proceso es que ha de ser lo más rápido posible. Por tanto, el método tradicional de analizar las páginas HTML como una secuencia, no conforma una buena solución. La productividad obtenida es baja: se obtienen pocas páginas en poco tiempo. Es de hecho, la peor solución. Entonces, para poder aumentar la respuesta del sistema, sólo nos queda una alternativa. Dividir el proceso en piezas más pequeñas, denominadas *tareas*, y ejecutarlas de forma concurrente. Esto nos lleva al uso de la programación multitarea o multihilo, confirmando que el anterior dicho, en ciertas circunstancias, es la mejor solución.

Con motivo de aumentar aún más el paralelismo, todos los *threads* o hilos se han diseñado para que tuvieran el mismo comportamiento. De este modo, todos luchan para obtener la CPU todo el tiempo. No hay hilos que sean ejecutados en un punto de ejecución concreto y el resto del tiempo permanezcan ociosos. Todo esto permite obtener una productividad *alta*. Ahora, en lugar de trabajar en serie, $P_{analysis}$ trabaja totalmente en paralelo. Esto es mostrado en la siguiente figura:

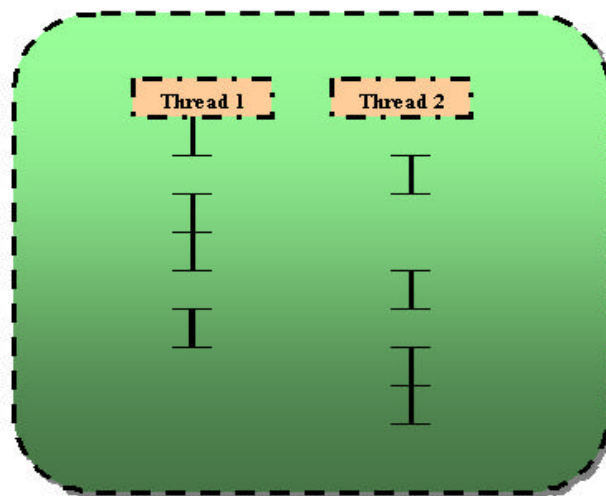


Figura 10. Muestra el nivel de paralelismo entre hilos con el mismo comportamiento.

Los hilos o *threads* son planificados por un planificador interno. Este controla las transiciones entre estados de los hilos involucrados en el proceso de análisis. También, se ocupa automáticamente de que a los hilos no les falte nuevas **URL** [**Uniform Resource Locator**, *identificador de recursos de la red*] para analizar. Si esto ocurre, entonces, él se ocupa de empujar a los hilos al estado de **ZOMBIE**.

Los hilos se han construido encima de la *JVM [Java Virtual Machine]*. Por tanto, el planificador se ha implementado usando las herramientas provistas por Java estándar y en particular, usando las primitivas de la clase *Object*: *wait()*, *notify()* etc.

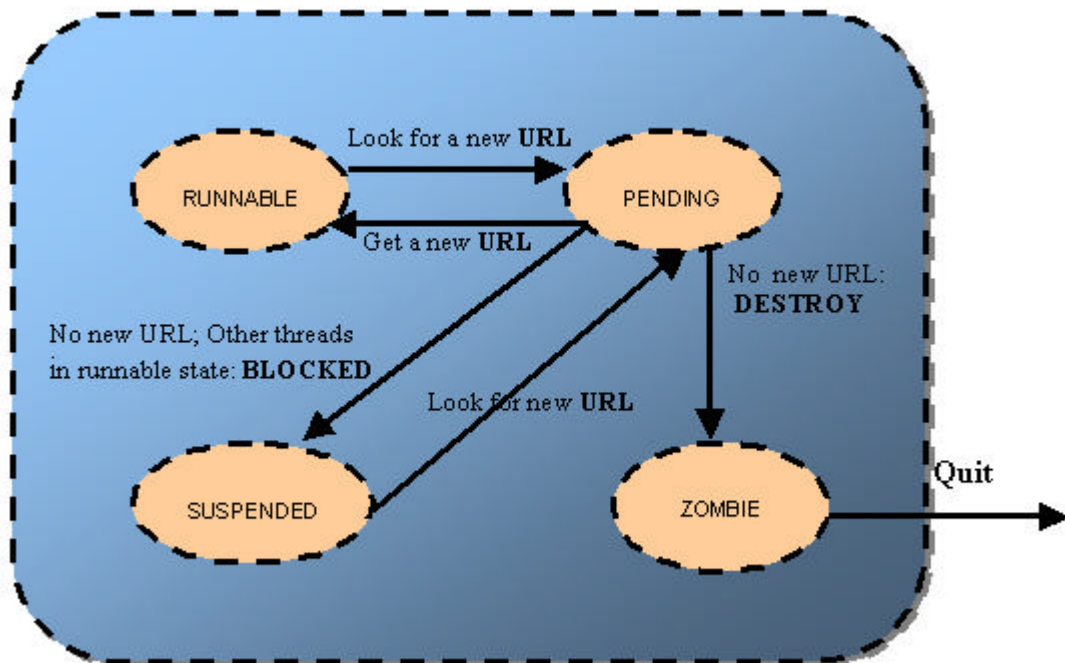


Figura 11. Muestra el ciclo de vida de los hilos.

Los estados en los cuales cualquier *thread* o hilo puede estar son los siguientes (tal como muestra la figura de más arriba):

- ❑ **RUNNABLE**: los threads están en ese estado siempre que lanzan el proceso de análisis sobre un nuevo recurso y por tanto, han obtenido una nueva referencia a un recurso de la red. Así pues, este estado se corresponde con un estado en el cual los hilos realizan trabajo útil. En este caso, *parsear* una página HTML y obtener nuevas referencias que son añadidas a la cola de recursos *por visitar*.
- ❑ **PENDING**: este estado es alcanzado por los hilos cuando intentan obtener una nueva URL. En este estado, cualquier *thread* intenta obtener una nueva referencia en primera instancia. Si no es posible, comprueba si existe algún otro hilo en estado de **RUNNABLE** (y por tanto, es susceptible de que añada nuevos recursos a la cola de recursos *por visitar*). Entonces, si esta última proposición es cierta, el hilo, en estado de **PENDING**, pasa a estado de **SUSPENDED** y se bloquea en un cola **FIFO**, esperando a que otro hilo lo despierte cuando URLs extras sean añadidas a la cola de recursos *por visitar*. En el caso de que la proposición anterior sea falsa, el hilo en cuestión pasa a estado de **ZOMBIE**. Si no hay ningún *thread* en estado de **RUNNABLE**, significa, que todos están esperando a que algún hilo añada nuevas referencias a la cola y éste fue, en realidad, su última oportunidad. Por tanto, una vez que el hilo obtiene este estado, es destruido, pero no sin antes despertar a los hilos bloqueados y dar parte de ello a la barrera. La barrera espera que todos los hilos finalicen antes de empezar a construir la *response ontology* para esta clase.

- ❑ **SUSPENDED:** Un *hilo* es suspendido cuando hay otros *hilos* que son susceptibles de añadir nuevas referencias (están en estado de **RUNNABLE**) pero, en este momento, no hay ninguna disponible en la cola de recursos *por visitar*. En este estado, el *hilo* es bloqueado esperando a que algún thread en estado de **RUNNABLE** lo despierte. Cuando lo haga, referencias extras habrán sido añadidas en la cola de recursos *por visitar*.
- ❑ **ZOMBIE:** un hilo obtiene este estado cuando no hay más recursos en la cola de recursos *por visitar*. Este estado ha sido diseñado para permitir realizar a los hilos operaciones de limpieza antes de finalizar, y, para evitar *interrumpir* su ciclo de vida. Esto es, para proveer un mecanismo fiable que preserve su “*muerte natural*”. De esta forma, cuando el *deadline* ha transcurrido, este estado, permite a todos los threads finalizar su última tarea antes de ser destruidos. Además, los hilos despiertan a los hilos durmientes (si aún quedan) y dan parte de su finalización a la barrera en este estado.

Los *hilos* están encapsulados en la clase **parser**. Esta clase implementa la interfaz *runnable*. Por tanto, nuevos hilos pueden ser instanciados a partir de ésta. Esta clase contiene varios métodos a parte del método *run()* obligatorio (el método donde la acción tiene lugar una vez que el hilo es iniciado). Estos métodos han sido diseñados para dotar de mayor funcionalidad al método *run()*. Estos métodos son: el constructor de la clase y el método responsable del proceso de obtención de nuevas URLs (*forma parte del planificador interno*). El constructor ha sido diseñado para instanciar todas las herramientas necesarias, como por ejemplo, el tipo de *parser* usado, y para preparar los objetos compartidos por todo los hilos. El constructor decide que *parser* será usado: o bien el *parser JTIDY*, o bien el *parser* creado a partir del estándar *JAVA*.

A continuación, exponemos el pseudocódigo del método *run()* (el corazón de todo hilo):

```
public void run()
{
    añadir el hilo a la barrera;

    mientras (el hilo no esté en estado de ZOMBIE) hacer
    {
        URL = obtener una URL nueva;
        si (el hilo está en estado de RUNNABLE) entonces
        {
            resource = crear un descriptor de recurso;
            fijar URL como la URL del descriptor resource;
            code = intentar parsear el recurso;
            si (code == éxito) entonces
            {
                actualizar el conjunto de páginas
                visitadas con el descriptor resource;
            }
        }
    }
}
```

```

    si (si hi hay hilos bloqueados(SUSPENDE)) entonces
    {
        despertarlos;
    }
    dar parte a la barrera de su finalización;
}

```

Una vez explicado el modelo computacional de $P_{analysis}$, pasaremos a detallar como este proceso analiza las páginas HTML.

El proceso $P_{analysis}$ analiza las páginas HTML puntuando las ocurrencias de la *ontología de búsqueda* [**query ontology**] encontradas en el texto. Es decir, comprueba para cadena de texto si existe un término en la *ontología* que coincida con él. Por cadena de texto, entendemos todas las palabras de un fichero HTML que son visibles por un usuario final en un navegador. Por tanto, este proceso ignora las coincidencias en el nombre de las marcas HTML o los atributos.

La puntuación de las ocurrencias tiene en cuenta dos factores:

- ❑ **Contexto:** es decir, las ocurrencias son puntuadas según en que conjunto de etiquetas están embebidas. Así, por ejemplo, un término que se encuentre contenido dentro de **** y **** (en negrita) tendrá mayor puntuación que un término en texto llano.
- ❑ **Categoría:** es decir, según si la ocurrencia coincide con un término *clase* o con un término *propiedad*. De esta forma, las páginas con términos que coinciden con nombres de *clases* tendrán mayor puntuación que las que sólo coincidan unas pocas *propiedades*.

El valor de cada etiqueta puntuable así como el peso de las dos categorías es configurable. De esta forma, $P_{analysis}$ puede ser configurado para analizar páginas según algún criterio del usuario. Así pues, un usuario podrá dar más relevancia a las propiedades que a las clases o valorar más el texto en *cursiva* que en *negrita*. El rango de valores de cualquier elemento configurable es de 0 a 30 puntos.

La puntuación final de cada página se obtiene sumando las puntuaciones de todas las ocurrencias encontradas en el texto. Esto implica que el valor final no esté normalizado. Para normalizar las páginas, se divide su puntuación por el valor de la página de máxima puntuación encontrada hasta el momento. De esta forma, se evita recalcular constantemente las puntuaciones de las páginas cuando el máximo actual es superado. Las páginas simplemente mantienen el valor normalizado en relación con el máximo absoluto en el momento de su puntuación. Sin embargo, esto puede parecer incoherente, puesto que podría darse el caso que la primera página con puntuación muy baja, al ser sí misma el único máximo, fuera normalizada a **1**. Sin embargo, como que partimos de *links* obtenidos en buscadores tales como **GOOGLE**, seguramente, la primera página tendrá una puntuación muy elevada. Esto impedirá que se hagan muchas recalculaciones y por tanto, las puntuaciones finales sean siempre muy fiables.

El uso del anterior esquema de *normalización* viene condicionado por el siguiente motivo: el filtrado. El proceso de filtrado o P_{filter} es el proceso que filtra las páginas analizadas según el umbral normalizado o **threshold** del usuario. Las páginas que no superan el umbral, son desechadas. Por motivos de eficiencia, el filtrado siempre se realiza a continuación del análisis de la página. Entonces, en ese mismo instante, necesitamos disponer ya de la puntuación normalizada para poder compararla con el *umbral*. No podemos esperar a encontrar la puntuación máxima y después dividir las páginas por este valor.

Una vez expuestos los detalles de su funcionamiento, pasaremos a explicar como se ha implementado el *analizador*.

El *analizador* ha de puntuar las páginas según el contexto y la categoría de los términos. Además, ha de ser rápido, de él depende la eficacia global del sistema. Entonces para evitar tener que hacer recorridos lineales $O(n)$, la obtención de la puntuación de una etiqueta y la comprobación de si una cadena de texto forma parte de la **query ontology**, se ha realizado a partir de tablas de **hash**. Esto permite obtener un coste constante $O(1)$ en ambas operaciones. Para entender esto, pongamos un ejemplo:

Supongamos que el *analizador (thread)* está analizando la siguiente línea de una página:

 RDF es el nuevo estándar de W3C para la web semántica

Supongamos además, que el valor de la etiqueta es 5 y que **RDF** es una ocurrencia de tipo clase de la **query ontology**.

Sin utilizar tablas de **hash**, el análisis de esta línea tendría un coste de: $O(n) * k + O(h)$. Siendo:

- n el número de términos en la **query ontology**.
- k el número de palabras de la línea
- h el número de etiquetas total puntuable.

Sin embargo, si utilizáramos una tabla de **hash** para almacenar los términos de la **query ontology**, podríamos comprobar directamente si una palabra es un término de la ontología. Simplemente, aplicaríamos la función de **hashing** sobre la palabra del texto y comprobaríamos si nos ha devuelto un acierto. De esta forma, se reduciría el coste de $O(n) * k$ a $O(1) * k$.

Además, si utilizáramos otra tabla de **hash** que almacenará las etiquetas, podríamos obtener la puntuación de cualquiera de ellas en coste constante $O(1)$. Así, para saber que está valorada con 5, simplemente, tendríamos que aplicar la función de **hashing** sobre el nombre de la etiqueta, ir a la posición y obtener su puntuación. Todo en coste constante $O(1)$. De esta forma, se reduciría el coste de $O(h)$ a $O(1)$.

He aquí la justificación de porqué el uso de las tablas de **hash**.

Para obtener la puntuación total del **contexto** cuando se encuentra un término, se ha utilizado una *pila*. Esta pila almacena las etiquetas que se han abierto pero que aún no se han cerrado. Es decir, sirve para manipular las etiquetas *anidadas*. Además, se ha utilizado una variable que almacena la suma total de todas las etiquetas abiertas. Así, cuando una palabra de la *ontología de búsqueda* es encontrada, simplemente, se devuelve el valor de esta variable (que es la puntuación total del contexto). La pila se usa para poder restar del **contexto** (de la variable) el valor de la última etiqueta abierta cuando esta se cierra. Así en todo momento, está disponible el valor total del contexto. No se ha de recalcular continuamente.

Ahora sólo falta mencionar que el *analizador* puede estar constituido por uno de los dos parsers siguientes: El construido mediante JTIDY o el construido mediante JAVA estándar. La única diferencia entre ambos es su mecanismo de trabajo. JTIDY funciona usando la aproximación **DOM**[10], mientras que el parser de JAVA actúa de forma similar a la aproximación **SAX**[11]. Estos parsers se han implementado para que el usuario pueda disponer de ambos tipos y puede elegir cuál de ellos le reporta mejores resultados.

6.2.3 Response Ontology

La response ontology se genera cuando el *deadline* ha transcurrido. El control del *deadline* se realiza usando un temporizador. Cuando un agente recibe una petición de búsqueda, su *ServerBehaviour* configura el temporizador con un retardo igual al *deadline* y empieza a ejecutar los *hilos*. Finalmente, cuando el tiempo ha transcurrido, el temporizador lanza la tarea que tiene asociada, en este caso, $P_{response}$.

$P_{response}$ se ocupa de generar la *response ontology*. Pero además, se ocupa de que todos los *hilos* finalicen poniéndolos en estado de **ZOMBIE**. De esta forma, la información necesaria para generar la *response ontology* estará cien por cien actualizada. Sólo cuando todos los *hilos* han dado parte de su finalización a la barrera y ésta se levanta, empieza la generación de la ontología. Finalmente, cuando ésta se ha generado, $P_{response}$ envía un mensaje de tipo INFORM al agente coordinador con la respuesta. Si habido problemas en su generación, $P_{response}$ envía un FAILURE en su lugar.

La *response ontology* está constituida por todas las URL provenientes del conjuntos de *recursos visitados* que superan el umbral. Además, cada URL aparece descrita por una serie de propiedades. Estas son:

- ❑ *Title*: el título de la página HTML.
- ❑ *Cite*: una cita significativa con palabras clave de la ontología de búsqueda.
- ❑ *Img*: el número de imágenes de la página HTML.
- ❑ *Weight*: la puntuación normalizada obtenida por la página.

A continuación, y para finalizar, incluimos un ejemplo de response ontology:

```

<?xml version='1.0'?>
<rdf:RDF
  xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:RDFNsld1='http://127.0.0.1:8080/examples/resource-ont#>

  <rdf:Description rdf:about='MotorVehicle'>
    <RDFNsld1:url>
      <rdf:Description rdf:about='http://127.0.0.1:8080/examples/index.html'
        RDFNsld1:title='THE MOTORVEHICLE IN THE WORLD'
        RDFNsld1:weight='1.0'
        RDFNsld1:img='0'>
    <RDFNsld1:cite>&lt;STRONG&gt;MOTORVEHICLE&lt;/STRONG&gt;
    </RDFNsld1:cite>
  </rdf:Description>
</RDFNsld1:url>
</rdf:Description>
</rdf:RDF>

```

6.3 Agente Coordinador(A_{coord})

En esta sección explicaremos como se ha definido el agente de Coordinador. Recordemos que la principal tarea de este agente es organizar a los agentes de *Internet* y elaborar una ontología con todos los resultados. Veamos como la realiza.

El agente de Coordinador realiza las siguientes tareas en su método *setup*:

- ❑ Registra sus servicios al DF de la plataforma: “CoordinatorAgent”.
- ❑ Inicializa su interfaz gráfico.
- ❑ Inicializa un objeto de tipo *readImpl*.
- ❑ Inicializa un objeto de tipo *writeImpl*.

El motivo de registrar el agente al *DF* es para indicar a otros agentes que servicios ofrece. Aunque en esta aplicación no es necesario registrar A_{coord} con *DF*, su registro se ha producido para dotar al sistema de mayor adaptabilidad . Si en un futuro este SMA es ampliado con más agentes, los nuevos agentes podrán conocer los servicios ofertados por agente a partir del agente *DF*.

La primera reflexión que deberíamos hacer sobre este agente es la siguiente: A_{coord} no ejecuta ninguna tarea o comportamiento al iniciarse. Es decir, permanece ocioso. La consecuencia directa es que este agente no consume CPU, y por tanto, permite una mayor optimización de los recursos de la plataforma. Sin embargo, nos preguntaremos ¿Cómo es capaz de empezar a realizar tarea alguna sin ningún comportamiento? La respuesta es bastante sencilla. Esta agente proporciona una interfaz gráfica a partir del cual el usuario puede indicarle que acciones debe llevar a cabo. Es decir, si debe extraer información o añadir / eliminar propiedades. Como todos estas opciones son gestionadas por eventos *AWT* / *SWING*(por ejemplo, al accionar un

menú), se ha decidido que sean las rutinas que capturan estos eventos las responsables de añadir los comportamientos de forma dinámica (mediante ***addBehaviour()***). De esta forma, los comportamientos son fruto de los eventos y permiten optimizar al máximo los recursos.

Una vez hecha esta pequeña reflexión, pasaremos a explicar como realiza las tareas este agente. Para facilitar la explicación, dividiremos el contenido en tres partes: en el proceso de división o $P_{splitter}$, en el proceso control o $P_{controller}$ y en el proceso de composición o $P_{assembler}$.

6.3.1 El proceso $P_{splitter}$

$P_{splitter}$ es el proceso que se ocupa de dividir los términos de ontología de entrada, en porciones individuales, que los $A_{internet}$ procesarán. La división de la ontología es por clases ***RDF***. Cada porción individual estará constituida por un recurso ***RDF*** de tipo ***clase*** o ***rdfs:Class*** y el conjunto de todas aquellas propiedades cuyo dominio de aplicación sea esta misma ***clase***. De esta forma, cada agente de Internet recibirá una ontología con una clase y sus propiedades [***query ontology***].

El objeto responsable de tal operación es ***readImpl***. Esta clase utiliza las facultades de ***JENA***, tales como ***sentencias RDQL***, para procesar y poder separar en clases la ontología de entrada o ***domain ontology***. Sin embargo, para que esta clase sea capaz de procesar esta ontología, son necesarios que cumpla una serie de requisitos. Estos son los siguientes:

- ❑ ***Estructura jerárquica***: debe seguir una estructura jerárquica correcta. Esto implica que cada clase debe estar correctamente incluida en su ***superclase*** mediante el uso de la propiedad ***rdfs:subClassOf***. Por ejemplo, si ***Truck*** es subclase de ***MotorVehicle*** su descripción sería la siguiente:

```
<rdf:Description rdf:about='Truck'>
  <rdfs:subClassOf rdf:resource='#MotorVehicle' />
</rdf:Description>
```

Para el elemento raíz de la domain ontology, la propiedad ***rdfs:subClassOf*** tomará como valor la clase ***rdfs:Resource***. Esto permitirá a ***readImpl*** distinguir la clase raíz de sus subclases.

Al igual, que los documentos ***XML***, la ***domain ontology*** sólo podrá presentar una estructura en forma de ***árbol***. Si un grafo es proporcionado, ***readImpl*** avisará de ello por medio de una ***excepción***.

- ❑ ***Dominio de las propiedades***: todas las propiedades deben incluir una o más veces la propiedad ***rdfs:domain***. Esta propiedad sirve para indicar sobre que clases o categorías se puede aplicar. Por ejemplo, si la

propiedad *registeredTo* se puede aplicar sobre *MotorVehicle* su descripción sería la siguiente.

```
<rdf:Description rdf:about='registeredTo'>
  <rdfs:domain rdf:resource='#MotorVehicle'/>
</rdf:Description>
```

El uso de esta propiedad permite a *readImpl* distribuir las propiedades por *clases*. Para entender esto supongamos el siguiente ejemplo:

Supongamos que una propiedad *x* es aplicable a dos clases. Entonces, en la *query ontology* de cada una de ellas, *x* habría de figurar como propiedad. Sin embargo, sin ningún tipo de indicación, *readImpl* no sería capaz de saber en que ontologías colocar la propiedad *x*, ¿En la dos? ¿En ninguna? etc. Se necesita utilizar alguna propiedad *RDF*, que permita conocer cuando una propiedad pertenece a un clase. Esta propiedad es ***rdfs:domain***.

- **Tipo del recurso:** todos los elementos descritos en la domain ontology han de ser tipificados o bien, como *rdfs:Class*(clases) o bien, como *rdfs:Property*(propiedades). Sin esta tipificación *readImpl* no es capaz de distinguir los recursos que son *propiedades* de los recursos que son *clases*. La especificación del tipo se hace por medio de ***rdf:type***.

Una vez que se han especificado los requisitos que ha de cumplir la ***domain ontology***, pasaremos a definir como funciona este proceso:

readImpl lee el fichero *RDF* con toda la ontología, comprueba que es un documento *RDF* bien formado y almacena su contenido en forma de ternas en memoria. Entonces, mediante sentencias ***RDQL***, divide el documento en porciones que quedan almacenadas en un *árbol*. Cada nodo corresponde con la definición de una *clase* y sus *propiedades*. Esta estructura de árbol se utiliza para conservar la estructura jerárquica cuando se componga, finalmente, la ontología de salida o ***information ontology***. Además, la división de la ***domain ontology*** en clases implica volver a generar los esquemas(*una propiedad puede ser compartida por varias clases*). Por tanto, es necesario almacenar las clases y las propiedades hasta volver a generar sus esquemas. Esto se produce cuando se envían definitivamente a los agentes de Internet.

El almacenamiento en forma de *árbol* de la ***domain ontology*** conlleva separar las clases, también, en forma de *árbol*. Partiendo de la clase raíz, se tratarán primero sus subclases, después la subclases de éstas etc. Es decir, se tratarán mediante un esquema de recorrido en *amplitud*. Esto implica utilizar tres tipos de sentencias *RDQL*:

```
SELECT ?x WHERE ( ?x, < RDF.type>, <RDFS.Class> ) ,
( ?x, <RDFS.subClassOf>, <RDFS.Resource> )
```

Esta sentencia se utiliza para seleccionar el elemento raíz de la *domain ontology*. Su traducción literal es “*selecciona aquellos recursos de tipo clase que sean subclase de rdfs:Resource*”. Como únicamente el elemento raíz es en principio subclase de *rdfs:Resource*, esta sentencia debería volver este elemento. Sin embargo, también como en el $A_{Internet}$, se ha de realizar un control adicional. Esta sentencia podría devolver más de un elemento raíz(*grafo*) o ninguno.

```
SELECT ?x WHERE ( ?x, <RDF.type>, <RDFS.Class>),
( ?x, <RDFS.subClassOf>, <parent>)
```

Esta sentencia se utiliza para seleccionar todas las subclases de la clase especificada por *<parent>*. Su traducción literal es “*selecciona aquellos recursos de tipo clase que sean subclase de <parent>*”. Esta sentencia se utiliza para ir seleccionando las clases en *amplitud*. Primero las subclases de un nivel. A continuación, las subclases de otro nivel hasta llegar a las *hojas*.

```
SELECT ?x WHERE ( ?x, <RDF.type>, <RDF.Property>),
( ?x, <RDFS.domain>, <class>);
```

Finalmente, esta sentencia se utiliza para seleccionar todas las propiedades de la clase *<class>*. Su traducción literal es “*selecciona aquellos recursos de tipo propiedad que sean aplicables a la clase <class>*”. Esta sentencia basa su funcionalidad en el uso de la propiedad ***rdfs:domain***, como hemos mencionado anteriormente.

Entonces, el proceso de división es como sigue:

Se selecciona el nodo raíz mediante la primera sentencia. A continuación, se añaden las subclases a una cola ***FIFO*** utilizando la segunda. Después, empieza una iteración hasta que la cola queda vacía. El cuerpo de la iteración realiza los siguientes pasos:

1. *coge* el primer elemento de la cola.
2. *obtiene* las subclases de este elemento(*segunda sentencia*).
3. *añade* las subclases a la cola *FIFO*.
4. *obtiene* las propiedades del elemento(*tercera sentencia*).
5. *añade* un nodo al árbol con el nombre del elemento(*clase*) y sus propiedades (*query ontology*).

De esta forma, se añaden los nodos de forma jerárquica o en *amplitud*.

6.3.2 El proceso $P_{controller}$

$P_{controller}$ es el proceso responsable de asignar las **query ontologies** a los agentes de Internet. Veamos como realiza dicha tarea:

El primer paso que realiza el A_{coord} consiste en preguntar al DF si existen agentes en la plataforma que realicen servicios de *Internet*. Esta pregunta se realiza mediante una instancia de la clase *ServiceDescription* de JADE. En la instancia, A_{coord} especifica que busca agentes de Internet :“*InternetAgent*”. A continuación, solicita a DF que devuelve una lista con los agentes de Internet. En la lista figurarán los AID de estos agentes. Esta operación bloquea la actividad de A_{coord} . Sin embargo, no hay ningún otro comportamiento en la cola de tareas del agente.

Una vez obtenidos los AID de los agentes de Internet, A_{coord} empieza a distribuir las **query ontology** entre ellos. Para generarlas, recorre el *árbol* y llama a *writeImpl* con la información de cada *nodo*. A continuación, elabora un mensaje FIPA-Request para cada una de las **query ontology**. Finalmente, manda los mensajes.

Para controlar el protocolo de interacción FIPA-Request y por tanto, para controlar que las tareas asignadas a los $A_{internet}$ se realizan de forma correcta, este agente utiliza una instancia de un comportamiento denominado *AchieveReInitiator*. Este comportamiento envía un mensaje y permite gestionar todos los estados del protocolo de interacción. La gestión de los estados es realizada mediante llamadas *callback* a una serie de rutinas . Estas se corresponden con cada estado posible del protocolo y puede ser sobrescritas. Así, *handleAgree* es llamada cuando un mensaje *Agree* llega del receptor. Así pues, para proporcionar control sobre el ejercicio de las tareas de los $A_{internet}$, se han sobrescrito cada una de las rutinas correspondientes a los estados. De esta forma, las interacciones entre los $A_{internet}$ y este agente, han quedado resueltas. El algoritmo de distribución ha sido el siguiente:

Cuando A_{coord} ha elaborado, mediante *writeImpl*, una **query ontology**, selecciona un AID de algún agente de Internet y elabora un mensaje tipo *Request*. A continuación, manda el mensaje vía un comportamiento *AchieveReInitiator* sobrescrito por nosotros. Entonces, pueden ocurrir dos cosas: puede o bien, recibir un mensaje positivo(Agree) o bien, recibir uno negativo(Refuse o NotUnderstood). Si recibe un Agree, A_{coord} apunta que la **query ontology** para la clase *X* está en proceso de búsqueda. Si recibe un mensaje negativo, escoge otro AID y vuelve a probar “*fortuna*”. Si vuelve a recibir un mensaje negativo, se apunta que la **query ontology** para la clase *X* no se podrá realizar y finaliza el comportamiento para esta *clase*. Finalmente, se queda esperando recibir un *Inform* o un *Failure* de los agentes de Internet de los cuales ha recibido un Agree. Estos últimos, mientras tanto, empiezan extraer información de la red. Finalmente, cuando A_{coord} recibe la respuesta de todos(tanto positiva(Inform) como negativa(Failure)), lanza el proceso de composición de la **information ontology**.

6.3.3 El proceso $P_{assembler}$

Cuando el agente coordinador lanza este proceso, los nodos del *árbol* (que han podido ser tratados) ya contienen la información actualizada de todos los recursos encontrados en la red. Es decir, se han actualizado con las ***response ontology*** de los Agentes de Internet . Este hecho es importante para el proceso de elaboración:

La elaboración de la ***information ontology***, simplemente, consiste en recorrer los nodos del *árbol* en *amplitud* concatenando su información. De esta forma, se mantiene la jerarquía de clases también en la *ontología*. Los recursos de cada nodo o *clase* antes de ser listados, son ordenados por puntuación mediante un algoritmo *QuickSort*. De esta forma, el usuario puede observar que páginas *HTML* han obtenido mayor puntuación de forma rápida. La representación de la ***information ontology***, una vez compuesta, es de dos formas: una es mediante una presentación gráfica en formato *HTML*, donde se listan las ocurrencias de mayor puntuación y la otra es mediante un fichero en formato *RDF*. Ejemplos de estas dos representaciones serán mostradas más adelante en el juego de pruebas.

7. Resultados

El objetivo de esta sección es demostrar que el proyecto desarrollado funciona correctamente. Para realizar tal comprobación, se han dividido las pruebas en dos categorías. En la primera, se han realizado pruebas de forma local en una computadora. En la segunda, en cambio, se han realizado las operaciones de forma distribuida.

Para realizar las pruebas de ambas categorías se ha utilizado la misma ontología de búsqueda, el mismo servidor(**Google**), el mismo tipo de *parser* (el creado a partir de **JAVA estándar**) y el mismo umbral(**0.5**). De esta forma, el lector podrá contrastar los resultados. Para obtener una visión más completa de éstos, se han utilizado dos tiempos o **deadlines**: de 1 minuto y de 5 minutos, respectivamente. Por tanto, las pruebas de ambas categorías mostrarán los resultados agrupados según estos tiempos.

Los resultados se representarán gráficamente mediante diagramas de barras y diagramas de sectores. Además, se añadirán ejemplos de la **information ontology** en algunos casos concretos.

La **domain ontology** usada para los juegos de pruebas ha sido:

```
<rdf:RDF xml:lang="en"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

  <rdf:Description rdf:ID="MAS">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-
      schema#Class"/>
    <rdfs:subClassOf
      rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
  </rdf:Description>

  <rdf:Description rdf:ID="ontology">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-
      schema#Class"/>
    <rdfs:subClassOf rdf:resource="#MAS"/>
  </rdf:Description>

  <rdf:Description rdf:ID="DAML">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-
      schema#Class"/>
    <rdfs:subClassOf rdf:resource="#ontology"/>
  </rdf:Description>

  <rdf:Description rdf:ID="RDF">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-
      schema#Class"/>
    <rdfs:subClassOf rdf:resource="#ontology"/>
  </rdf:Description>

  <rdf:Description rdf:ID="conferences">
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-
      syntax-ns#Property"/>
    <rdfs:domain rdf:resource="#MAS"/>
    <rdfs:domain rdf:resource="#ontology"/>
  </rdf:Description>
```

```

<rdf:Description rdf:ID="publications">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-
syntax-ns#Property"/>
  <rdfs:domain rdf:resource="#RDF"/>
  <rdfs:domain rdf:resource="#DAML"/>
  <rdfs:domain rdf:resource="#ontology"/>
</rdf:Description>

<rdf:Description rdf:ID="theory">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-
syntax-ns#Property"/>
  <rdfs:domain rdf:resource="#RDF"/>
  <rdfs:domain rdf:resource="#DAML"/>
  <rdfs:domain rdf:resource="#ontology"/>
  <rdfs:domain rdf:resource="#MAS"/>
</rdf:Description>

</rdf:RDF>

```

Esquemáticamente sería:

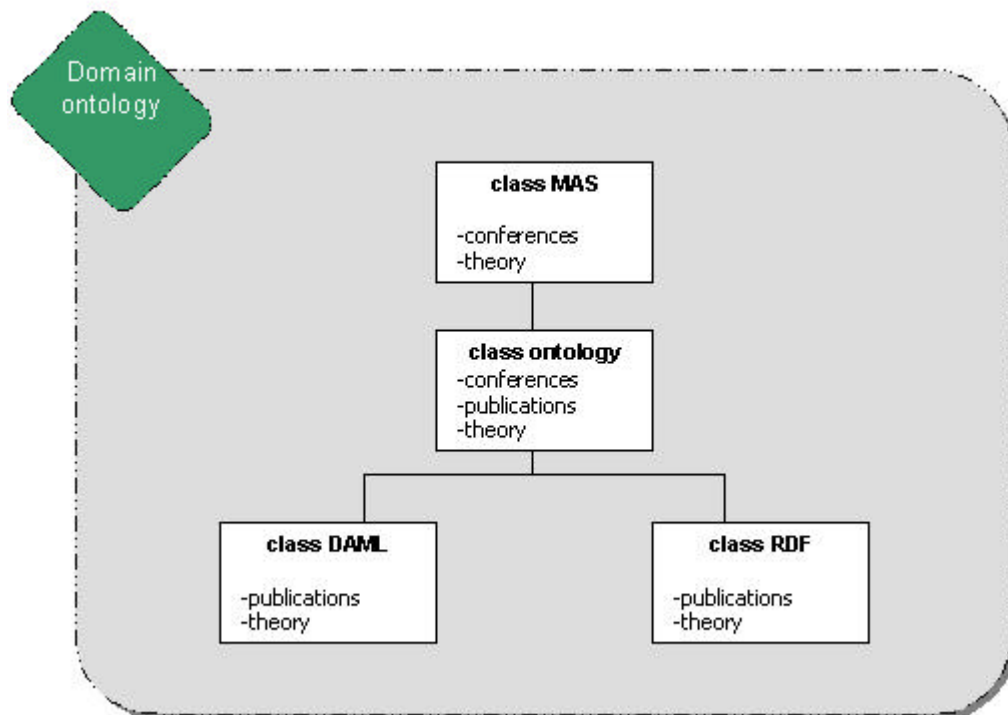


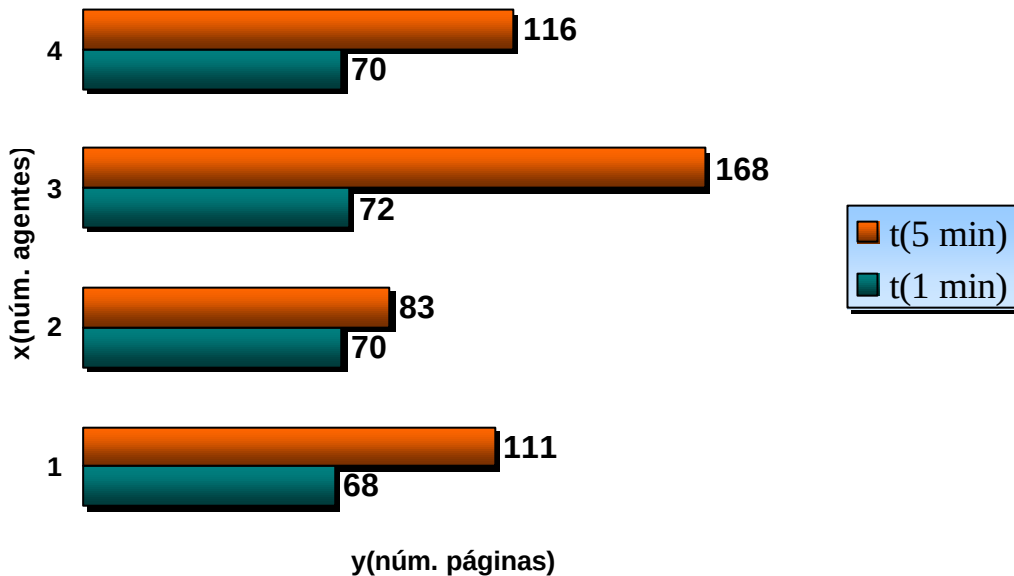
Figura 12. Muestra en forma de esquema la ontología de entrada del sistema.

Una vez comentadas las condiciones de ambos juegos de pruebas, vamos a empezar a mostrar sus resultados. En primer lugar, mostraremos los resultados para las pruebas locales. En segundo lugar, mostraremos las pruebas para las pruebas distribuidas.

Juego de pruebas Local

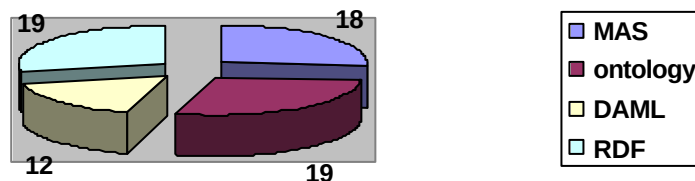
La realización de este juego de pruebas se realizó partiendo de un sólo $A_{internet}$ hasta un máximo de cuatro. Colocar más agentes no tendría sentido puesto que sólo hay cuatro categorías (cada $A_{internet}$ tendrá entonces una categoría).

Para contrastar el número de páginas encontradas respecto a 1 minuto de tiempo o a 5 minutos de tiempo se ha utilizado un diagrama de barras. En él se presentan los resultados agrupados según el número de agentes utilizado en aquel momento en la plataforma. Esta gráfica se muestra a continuación:

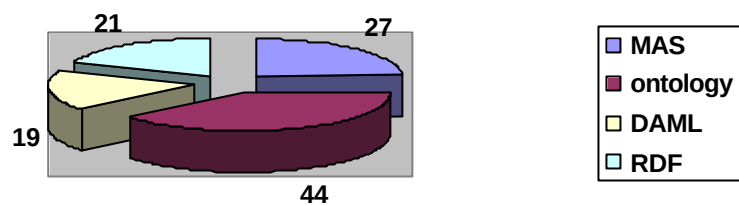


Como podemos observar en la anterior gráfica, aumentar el número de agentes en la plataforma no mejora substancialmente los resultados. La conexión hace de cuello de botella y las primeras páginas encontradas a partir de **Google** suelen ser siempre las de mayor puntuación. A continuación exponemos las gráficas según sus clases de representación:

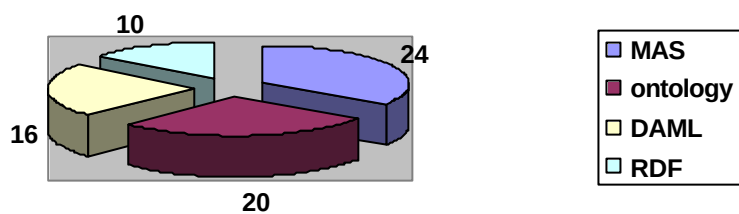
Distribución de clases para 1 agente en 1 minuto



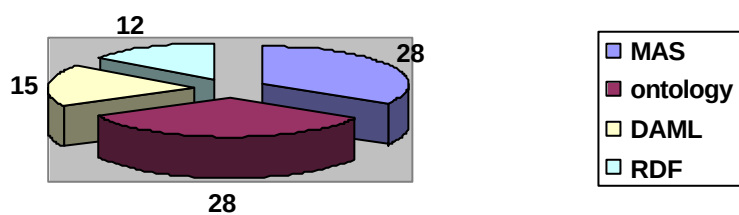
Distribución de clases para 1 agente en 5 minutos



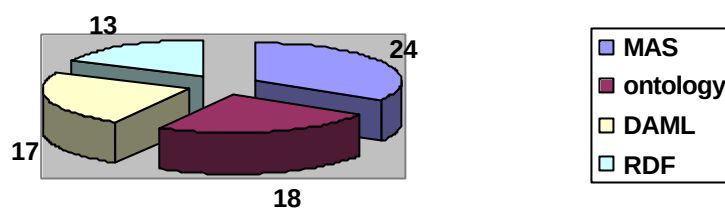
Distribución de clases para 2 agentes en 1 minuto



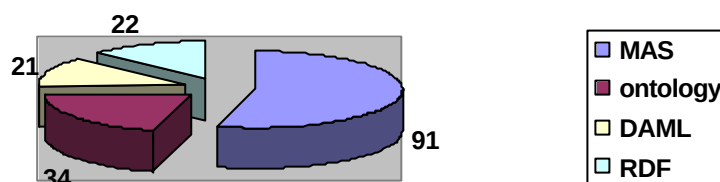
Distribución de clases para 2 agentes en 5 minutos



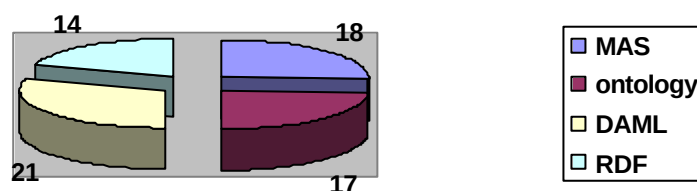
Distribución de clases para 3 agentes en 1 minuto



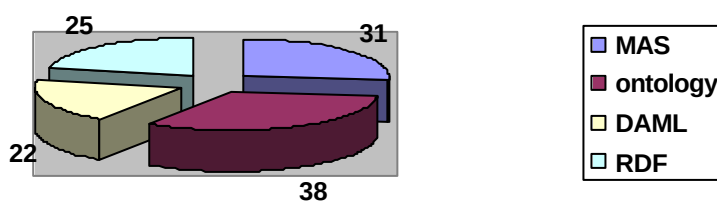
Distribución de clases para 3 agentes en 5 minuto



Distribución de clases para 4 agentes en 1 minuto



Distribución de clases para 4 agente en 5 minutos



Para comprobar la adición y eliminación de propiedades de forma dinámica se ha realizado el siguiente juego de pruebas:

Se realizado una nueva búsqueda con tres agentes en 5 minutos, pero, quitando la propiedad **publications** de sus clases de forma dinámica en mitad de la búsqueda. Estas clases son **RDF**, **DAML** y **ontology**. A continuación, se han contrastado los resultados con el juego de pruebas local de 3 agentes en 5 minutos. Una vez contrastados, se ha podido observar que algunas páginas que poseían la palabra **publications** como cita aparecen en de nuevo en esta última búsqueda pero sin esta palabra en ella. Esto demuestra que para algunas páginas la propiedad **PUBLICATIONS** no se ha tenido en cuenta. También, se puede demostrar de forma indirecta porque en esta última búsqueda el número de páginas ha disminuido.

Pongamos un ejemplo:

En la clase **DAML** la siguiente página ha aparecido en las dos búsquedas:

```
DAML PUBLICATIONS http://www.daml.org/
  Title: DAML.org
  Images: 2
  Ratio: 1

DAML http://www.daml.org
  Title: DAML.org
  Images: 2
  Ratio: 1
```

Pero en la última podemos observar que **PUBLICATIONS** ha caído de la cita.

Finalmente, exponemos un ejemplo de representación de la **information ontology** en **RDF** correspondientes al caso de 2 agentes en 1 minuto.

```
<?xml version='1.0'?>
<rdf:RDF
  xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:RDFNsId1='http://127.0.0.1:8080/examples/resource-ont#'
  xmlns:rdfs='http://www.w3.org/2000/01/rdf-schema#'>
  <rdf:Description rdf:about='Information Ontology'>
    <RDFNsId1:class>
      <rdf:Description rdf:about='MAS'>
        <rdfs:subClassOf
rdf:resource='http://www.w3.org/2000/01/rdf-schema#Resource' />
        <RDFNsId1:url>
          <rdf:Description
rdf:about='http://www.unomas.com/reviews/music/reviews/item096.html'
          RDFNsId1:weight='1'
          RDFNsId1:img='17'>
            <RDFNsId1:title>UNo MAS: Alejandro Escovedo,
Man Under the Influence</RDFNsId1:title>
            <RDFNsId1:cite>UNO
&lt;STRONG&gt;MAS&lt;/STRONG&gt; MAGAZINE </RDFNsId1:cite>
            </rdf:Description>
          </RDFNsId1:url>
        </RDFNsId1:url>
      </rdf:Description>
    </RDFNsId1:class>
  </rdf:Description>
</rdf:RDF>
```

```

        <rdf:Description
rdf:about='http://www.unomas.com/reviews/film/reviews/item004.html'
        RDFNsId1:weight='1'
        RDFNsId1:img='17'>
        <RDFNsId1:title>UNo MAS: Realm of the Senses -
1976, Nagisa Oshima</RDFNsId1:title>
        <RDFNsId1:cite>UNO
&lt;STRONG&gt;MAS&lt;/STRONG&gt; MAGAZINE </RDFNsId1:cite>
        </rdf:Description>
        </RDFNsId1:url>
        <RDFNsId1:url>
        <rdf:Description
rdf:about='http://www.unomas.com/reviews/music/reviews/item003.html'
        RDFNsId1:weight='1'
        RDFNsId1:img='17'>
        <RDFNsId1:title>UNo MAS: Pavement - Brighten
the Corners</RDFNsId1:title>
        <RDFNsId1:cite>UNO
&lt;STRONG&gt;MAS&lt;/STRONG&gt; MAGAZINE </RDFNsId1:cite>
        </rdf:Description>
        </RDFNsId1:url>
        <RDFNsId1:url>
        <rdf:Description
rdf:about='http://www.unomas.com/reviews/film/reviews/item003.html'
        RDFNsId1:weight='1'
        RDFNsId1:img='17'>
        <RDFNsId1:title>UNo MAS: Vincent and Theo -
1990, Robert Altman</RDFNsId1:title>
        <RDFNsId1:cite>UNO
&lt;STRONG&gt;MAS&lt;/STRONG&gt; MAGAZINE </RDFNsId1:cite>
        </rdf:Description>
        </RDFNsId1:url>
        <RDFNsId1:url>
        <rdf:Description
rdf:about='http://www.unomas.com/reviews/book/reviews/item017.html'
        RDFNsId1:weight='1'
        RDFNsId1:img='17'>
        <RDFNsId1:title>UNo MAS: Say Goodbye - Lewis
Shiner</RDFNsId1:title>
        <RDFNsId1:cite>UNO
&lt;STRONG&gt;MAS&lt;/STRONG&gt; MAGAZINE </RDFNsId1:cite>
        </rdf:Description>
        </RDFNsId1:url>
        <RDFNsId1:url>
        <rdf:Description
rdf:about='http://www.unomas.com/reviews/music/reviews/item094.html'
        RDFNsId1:weight='1'
        RDFNsId1:img='17'>
        <RDFNsId1:title>UNo MAS: Badlands - A Tribute
to Bruce Springsteen's Nebraska</RDFNsId1:title>
        <RDFNsId1:cite>UNO
&lt;STRONG&gt;MAS&lt;/STRONG&gt; MAGAZINE </RDFNsId1:cite>
        </rdf:Description>
        </RDFNsId1:url>
        <RDFNsId1:url>
        <rdf:Description
rdf:about='http://www.unomas.com/reviews/film/reviews/item002.html'
        RDFNsId1:weight='1'
        RDFNsId1:img='17'>

```

<RDFNSId1:title>UNo MAS: The Big Bird Cage,
 Coffy, Foxy Brown - Written and Directed by Jack Hill (1973 and
 1974)</RDFNSId1:title>
 <RDFNSId1:cite>UNO
 MAS MAGAZINE </RDFNSId1:cite>
 </rdf:Description>
 </RDFNSId1:url>
 <RDFNSId1:url>
 <rdf:Description
 rdf:about='http://www.unomas.com/reviews/book/reviews/item016.html'
 RDFNSId1:weight='1'
 RDFNSId1:img='17'>
 <RDFNSId1:title>UNo MAS: A Book of Memories -
 Péter Nádas (Trans. by Ivan Sanders and Imre
 Goldstein)</RDFNSId1:title>
 <RDFNSId1:cite>UNO
 MAS MAGAZINE </RDFNSId1:cite>
 </rdf:Description>
 </RDFNSId1:url>
 <RDFNSId1:url>
 <rdf:Description
 rdf:about='http://www.unomas.com/reviews/music/reviews/item001.html'
 RDFNSId1:weight='1'
 RDFNSId1:img='17'>
 <RDFNSId1:title>UNo MAS: Nashville: The Other
 Side of the Alley - Insurgent Country Volume 3</RDFNSId1:title>
 <RDFNSId1:cite>UNO
 MAS MAGAZINE </RDFNSId1:cite>
 </rdf:Description>
 </RDFNSId1:url>
 <RDFNSId1:url>
 <rdf:Description rdf:about='http://www.mas.ru/'
 RDFNSId1:weight='1'
 RDFNSId1:img='50'>
 <RDFNSId1:title>MAS Elektronik
 AG</RDFNSId1:title>
 <RDFNSId1:cite>ÊÏÏÏÀÍËÿ
 MAS</RDFNSId1:cite>
 </rdf:Description>
 </RDFNSId1:url>
 <RDFNSId1:url>
 <rdf:Description
 rdf:about='http://www.unomas.com/reviews/music/reviews/item093.html'
 RDFNSId1:weight='1'
 RDFNSId1:img='17'>
 <RDFNSId1:title>UNo MAS: The Gourds - Bolsa de
 Agua</RDFNSId1:title>
 <RDFNSId1:cite>UNO
 MAS MAGAZINE </RDFNSId1:cite>
 </rdf:Description>
 </RDFNSId1:url>
 <RDFNSId1:url>
 <rdf:Description
 rdf:about='http://greg.turtleprod.com/pictures/'
 RDFNSId1:weight='1'
 RDFNSId1:img='4'>
 <RDFNSId1:title>Greg's Home Space:
 Pictures</RDFNSId1:title>
 <RDFNSId1:cite>UNO
 MAS </RDFNSId1:cite>
 </rdf:Description>

```

        </RDFNSId1:url>
        <RDFNSId1:url>
        <rdf:Description
rdf:about='http://www.unomas.com/reviews/book/reviews/item001.html'
        RDFNSId1:weight='1'
        RDFNSId1:img='17'>
        <RDFNSId1:title>UNo MAS: Age and Guile Beat
Youth, Innocence, and a Bad Haircut: 25 Years of P.J. O'&Rourke -
P.J. O'&Rourke</RDFNSId1:title>
        <RDFNSId1:cite>UNO
&lt;STRONG&gt;MAS&lt;/STRONG&gt; MAGAZINE </RDFNSId1:cite>
        </rdf:Description>
        </RDFNSId1:url>
        <RDFNSId1:url>
        <rdf:Description
rdf:about='http://www.unomas.com/reviews/book/reviews/item014.html'
        RDFNSId1:weight='1'
        RDFNSId1:img='17'>
        <RDFNSId1:title>UNo MAS: She Loves Me - Peter
Esterhazy, trans. Judith Sollosy</RDFNSId1:title>
        <RDFNSId1:cite>UNO
&lt;STRONG&gt;MAS&lt;/STRONG&gt; MAGAZINE </RDFNSId1:cite>
        </rdf:Description>
        </RDFNSId1:url>
        <RDFNSId1:url>
        <rdf:Description
rdf:about='http://www.turtleprod.com/greg/'
        RDFNSId1:weight='1'
        RDFNSId1:img='4'>
        <RDFNSId1:title>Greg's Home Space:
Home</RDFNSId1:title>
        <RDFNSId1:cite>UNO
&lt;STRONG&gt;MAS&lt;/STRONG&gt; </RDFNSId1:cite>
        </rdf:Description>
        </RDFNSId1:url>
        <RDFNSId1:url>
        <rdf:Description
rdf:about='http://greg.turtleprod.com/frontier/'
        RDFNSId1:weight='1'
        RDFNSId1:img='4'>
        <RDFNSId1:title>Greg's Home Space:
Frontier Stuff</RDFNSId1:title>
        <RDFNSId1:cite>UNO
&lt;STRONG&gt;MAS&lt;/STRONG&gt; </RDFNSId1:cite>
        </rdf:Description>
        </RDFNSId1:url>
        <RDFNSId1:url>
        <rdf:Description
rdf:about='http://www.unomas.com/reviews/music/reviews/item099.html'
        RDFNSId1:weight='1'
        RDFNSId1:img='17'>
        <RDFNSId1:title>UNo MAS: T. Griffin, Light in
the Aisles</RDFNSId1:title>
        <RDFNSId1:cite>UNO
&lt;STRONG&gt;MAS&lt;/STRONG&gt; MAGAZINE </RDFNSId1:cite>
        </rdf:Description>
        </RDFNSId1:url>
        <RDFNSId1:url>
        <rdf:Description
rdf:about='http://www.unomas.com/reviews/music/reviews/item098.html'
        RDFNSId1:weight='1'

```

```

        RDFNsid1:img='17'>
        <RDFNsid1:title>UNo MAS: Old 97&apos;s,
Satellite Rides</RDFNsid1:title>
        <RDFNsid1:cite>UNO
&lt;STRONG&gt;MAS&lt;/STRONG&gt; MAGAZINE </RDFNsid1:cite>
        </rdf:Description>
        </RDFNsid1:url>
        <RDFNsid1:url>
        <rdf:Description
rdf:about='http://www.unomas.com/reviews/music/reviews/item097.html'
        RDFNsid1:weight='1'
        RDFNsid1:img='17'>
        <RDFNsid1:title>UNo MAS: Burning Airlines,
Identikit</RDFNsid1:title>
        <RDFNsid1:cite>UNO
&lt;STRONG&gt;MAS&lt;/STRONG&gt; MAGAZINE </RDFNsid1:cite>
        </rdf:Description>
        </RDFNsid1:url>
        <RDFNsid1:url>
        <rdf:Description
rdf:about='http://www.unomas.com/'
        RDFNsid1:weight='1'
        RDFNsid1:img='13'>
        <RDFNsid1:title>UNo MAS: Home</RDFNsid1:title>
        <RDFNsid1:cite>UNO
&lt;STRONG&gt;MAS&lt;/STRONG&gt; MAGAZINE </RDFNsid1:cite>
        </rdf:Description>
        </RDFNsid1:url>
        <RDFNsid1:url>
        <rdf:Description
rdf:about='http://www.unomas.com/reviews/film/reviews/item005.html'
        RDFNsid1:weight='1'
        RDFNsid1:img='17'>
        <RDFNsid1:title>UNo MAS: T-Men, Raw Deal -
Dir. Anthony Mann</RDFNsid1:title>
        <RDFNsid1:cite>UNO
&lt;STRONG&gt;MAS&lt;/STRONG&gt; MAGAZINE </RDFNsid1:cite>
        </rdf:Description>
        </RDFNsid1:url>
        <RDFNsid1:url>
        <rdf:Description
rdf:about='http://www.unomas.com/reviews/book/reviews/item002.html'
        RDFNsid1:weight='0,83'
        RDFNsid1:img='17'>
        <RDFNsid1:title>UNo MAS: Close Encounters of
the Fourth Kind - C.D.B. Bryan</RDFNsid1:title>
        <RDFNsid1:cite>UNO
&lt;STRONG&gt;MAS&lt;/STRONG&gt; MAGAZINE </RDFNsid1:cite>
        </rdf:Description>
        </RDFNsid1:url>
        <RDFNsid1:url>
        <rdf:Description
rdf:about='http://www.unomas.com/reviews/book/reviews/item013.html'
        RDFNsid1:weight='0,83'
        RDFNsid1:img='17'>
        <RDFNsid1:title>UNo MAS: Gut Symmetries -
Jeanette Winterson</RDFNsid1:title>
        <RDFNsid1:cite>UNO
&lt;STRONG&gt;MAS&lt;/STRONG&gt; MAGAZINE </RDFNsid1:cite>
        </rdf:Description>
        </RDFNsid1:url>

```

```

        <RDFNsId1:url>
        <rdf:Description
rdf:about='http://www.unomas.com/reviews/music/reviews/item002.html'
        RDFNsId1:weight='0,83'
        RDFNsId1:img='17'>
        <RDFNsId1:title>UNo MAS: Billy Bragg - William
Bloke</RDFNsId1:title>
        <RDFNsId1:cite>UNO
&lt;STRONG&gt;MAS&lt;/STRONG&gt; MAGAZINE </RDFNsId1:cite>
        </rdf:Description>
        </RDFNsId1:url>
    </rdf:Description>
</RDFNsId1:class>
<RDFNsId1:class>
    <rdf:Description rdf:about='ontology'>
        <rdfs:subClassOf rdf:resource='MAS' />
        <RDFNsId1:url>
        <rdf:Description
rdf:about='http://www.ksl.stanford.edu/projects/rkf/'
        RDFNsId1:weight='1'
        RDFNsId1:img='1'>
        <RDFNsId1:title>Rapid Knowledge Formation
Project for Stanford Knowledge Systems Laboratory</RDFNsId1:title>
        <RDFNsId1:cite>CHIMAERA
&lt;STRONG&gt;ONTOLOGY&lt;/STRONG&gt; ENVIRONMENT </RDFNsId1:cite>
        <RDFNsId1:weight>0,75</RDFNsId1:weight>

<RDFNsId1:cite>&lt;STRONG&gt;THEORY&lt;/STRONG&gt; MANIPULATION
</RDFNsId1:cite>
        </rdf:Description>
        </RDFNsId1:url>
        <RDFNsId1:url>
        <rdf:Description rdf:about='http://km.aifb.uni-
karlsruhe.de/owl/index.html'
        RDFNsId1:weight='1'
        RDFNsId1:img='2'>
        <RDFNsId1:title>Web Ontology
Requirements</RDFNsId1:title>
        <RDFNsId1:cite>THIS DOCUMENT SPECIFIES USEAGE
SCENARIOS, GOALS AND REQUIREMENTS FOR A WEB
&lt;STRONG&gt;ONTOLOGY&lt;/STRONG&gt;</RDFNsId1:cite>
        </rdf:Description>
        </RDFNsId1:url>
        <RDFNsId1:url>
        <rdf:Description
rdf:about='http://www.w3.org/2001/sw/WebOnt/'
        RDFNsId1:weight='1'
        RDFNsId1:img='3'>
        <RDFNsId1:title>W3C Web Ontology (WebOnt)
Working Group (OWL)</RDFNsId1:title>
        <RDFNsId1:cite>W3C WEB
&lt;STRONG&gt;ONTOLOGY&lt;/STRONG&gt; (WEBONT) WORKING GROUP (OWL)
</RDFNsId1:cite>
        </rdf:Description>
        </RDFNsId1:url>
        <RDFNsId1:url>
        <rdf:Description
rdf:about='http://www.ontology.org/'
        RDFNsId1:weight='1'
        RDFNsId1:img='13'>

```

```

                                <RDFNSId1:title>Ontology.Org - Enabling
Virtual Business</RDFNSId1:title>
                                <RDFNSId1:cite>SHARED
&lt;STRONG&gt;ONTOLOGY&lt;/STRONG&gt; </RDFNSId1:cite>
                                </rdf:Description>
                                </RDFNSId1:url>
                                <RDFNSId1:url>
                                <rdf:Description
rdf:about='http://www.formalontology.it/'
                                RDFNSId1:weight='1'
                                RDFNSId1:img='5'>
                                <RDFNSId1:title>Ontology - Descriptive and
Formal</RDFNSId1:title>
                                <RDFNSId1:cite>CONTEMPORARY
&lt;STRONG&gt;ONTOLOGY&lt;/STRONG&gt; </RDFNSId1:cite>
                                </rdf:Description>
                                </RDFNSId1:url>
                                <RDFNSId1:url>
                                <rdf:Description rdf:about='http://www-
ksl.stanford.edu/kst/what-is-an-ontology.html'
                                RDFNSId1:weight='1'
                                RDFNSId1:img='0'>
                                <RDFNSId1:title>What is an
Ontology?</RDFNSId1:title>
                                <RDFNSId1:cite>. THAT IS, AN
&lt;STRONG&gt;ONTOLOGY&lt;/STRONG&gt; </RDFNSId1:cite>
                                </rdf:Description>
                                </RDFNSId1:url>
                                <RDFNSId1:url>
                                <rdf:Description
rdf:about='http://www.formalontology.it/section_4.htm'
                                RDFNSId1:weight='1'
                                RDFNSId1:img='2'>
                                <RDFNSId1:title>What is Ontology? Definitions
by leading philosophers</RDFNSId1:title>
                                <RDFNSId1:cite>&quot;IN CONTEMPORARY
PHILOSOPHY, FORMAL
&lt;STRONG&gt;ONTOLOGY&lt;/STRONG&gt; </RDFNSId1:cite>
                                </rdf:Description>
                                </RDFNSId1:url>
                                <RDFNSId1:url>
                                <rdf:Description
rdf:about='http://www.formalontology.it/copernic_report.htm'
                                RDFNSId1:weight='1'
                                RDFNSId1:img='22'>
                                <RDFNSId1:title>Copernic Report on
Ontology</RDFNSId1:title>
                                <RDFNSId1:cite>... GENE
&lt;STRONG&gt;ONTOLOGY&lt;/STRONG&gt; </RDFNSId1:cite>
                                </rdf:Description>
                                </RDFNSId1:url>
                                <RDFNSId1:url>
                                <rdf:Description
rdf:about='http://www.w3.org/2001/sw/webont/webont-issues.html'
                                RDFNSId1:weight='1'
                                RDFNSId1:img='1'>
                                <RDFNSId1:title>Web Ontology
Issues</RDFNSId1:title>
                                <RDFNSId1:cite>THE WEB
&lt;STRONG&gt;ONTOLOGY&lt;/STRONG&gt; </RDFNSId1:cite>
                                </rdf:Description>

```

```

        </RDFNSId1:url>
        <RDFNSId1:url>
        <rdf:Description
rdf:about='http://www.formalontology.it/onto_papers.htm'
        RDFNSId1:weight='1'
        RDFNSId1:img='10'>
        <RDFNSId1:title>A selection of papers on
Ontology and Formal Ontology</RDFNSId1:title>
        <RDFNSId1:cite>A SELECTION OF PAPERS ON
&lt;STRONG&gt;ONTOLOGY&lt;/STRONG&gt; AND FORMAL
&lt;STRONG&gt;ONTOLOGY&lt;/STRONG&gt; </RDFNSId1:cite>
        </rdf:Description>
        </RDFNSId1:url>
        <RDFNSId1:url>
        <rdf:Description
rdf:about='http://www.geneontology.org/'
        RDFNSId1:weight='1'
        RDFNSId1:img='39'>
        <RDFNSId1:title>Gene Ontology
Consortium</RDFNSId1:title>
        <RDFNSId1:cite>GENE
&lt;STRONG&gt;ONTOLOGY&lt;/STRONG&gt; CONSORTIUM </RDFNSId1:cite>
        </rdf:Description>
        </RDFNSId1:url>
        <RDFNSId1:url>
        <rdf:Description
rdf:about='http://www.formalontology.it/section_5.htm'
        RDFNSId1:weight='1'
        RDFNSId1:img='2'>
        <RDFNSId1:title>Proposals about the Structure
of the Ontology</RDFNSId1:title>
        <RDFNSId1:cite>THE MAIN DISTINCTION IS BETWEEN
SUBSTANCE AND DETERMINATION. THE
&lt;STRONG&gt;THEORY&lt;/STRONG&gt;</RDFNSId1:cite>
        </rdf:Description>
        </RDFNSId1:url>
        <RDFNSId1:url>
        <rdf:Description
rdf:about='http://www.formalontology.it/link_books.htm'
        RDFNSId1:weight='0,76'
        RDFNSId1:img='2'>
        <RDFNSId1:title>Recent books on Formal
Ontology</RDFNSId1:title>
        <RDFNSId1:cite>RECENT BOOKS ON FORMAL
&lt;STRONG&gt;ONTOLOGY&lt;/STRONG&gt; AND THE
&lt;STRONG&gt;THEORY&lt;/STRONG&gt; OF OBJECTS </RDFNSId1:cite>
        </rdf:Description>
        </RDFNSId1:url>
        <RDFNSId1:url>
        <rdf:Description rdf:about='http://www-
db.research.bell-labs.com/user/pfps/owl/specification.html'
        RDFNSId1:weight='0,74'
        RDFNSId1:img='0'>
        <RDFNSId1:title>OWL Web Ontology Language 1.0
Abstract Syntax</RDFNSId1:title>
        <RDFNSId1:cite>THE OWL WEB
&lt;STRONG&gt;ONTOLOGY&lt;/STRONG&gt;</RDFNSId1:cite>
        </rdf:Description>
        </RDFNSId1:url>
        <RDFNSId1:url>

```

```

        <rdf:Description
rdf:about='http://www.geneontology.org/doc/go.curator_requests.html'
        RDFNsId1:weight='0,72'
        RDFNsId1:img='2'>
        <RDFNsId1:title>GO Curator Requests on
SourceForge.net</RDFNsId1:title>
        <RDFNsId1:cite>GENE
&lt;STRONG&gt;ONTOLOGY&lt;/STRONG&gt; HOME </RDFNsId1:cite>
        </rdf:Description>
        </RDFNsId1:url>
        <RDFNsId1:url>
        <rdf:Description
rdf:about='http://www.w3.org/2001/sw/webont/ftf3.html'
        RDFNsId1:weight='0,68'
        RDFNsId1:img='0'>
        <RDFNsId1:title>Third Meeting of the W3C Web
Ontology Working Group, Jul 2002 in Stanford, CA, USA</RDFNsId1:title>
        <RDFNsId1:cite>THIRD MEETING OF THE W3C WEB
&lt;STRONG&gt;ONTOLOGY&lt;/STRONG&gt; WORKING GROUP, JUL 2002 IN
STANFORD, CA, USA </RDFNsId1:cite>
        </rdf:Description>
        </RDFNsId1:url>
        <RDFNsId1:url>
        <rdf:Description
rdf:about='http://www.geneontology.org/doc/go_contacts.html'
        RDFNsId1:weight='0,65'
        RDFNsId1:img='19'>
        <RDFNsId1:title>How to Contact
GO</RDFNsId1:title>
        <RDFNsId1:cite>TO FOLLOW THE DEVELOPMENT OF
THE GENE &lt;STRONG&gt;ONTOLOGY&lt;/STRONG&gt; PROJECT, SEVERAL
</RDFNsId1:cite>
        </rdf:Description>
        </RDFNsId1:url>
        <RDFNsId1:url>
        <rdf:Description
rdf:about='http://www.formalontology.it/what.htm'
        RDFNsId1:weight='0,65'
        RDFNsId1:img='2'>
        <RDFNsId1:title>What&apos;s new on this
site</RDFNsId1:title>
        <RDFNsId1:cite>MAY, 27TH 2001: ADDITIONS TO
THE PAGE PAGES ON THE HISTORY OF
&lt;STRONG&gt;ONTOLOGY&lt;/STRONG&gt;</RDFNsId1:cite>
        </rdf:Description>
        </RDFNsId1:url>
        <RDFNsId1:url>
        <rdf:Description
rdf:about='http://www.ksl.stanford.edu/projects/daml/'
        RDFNsId1:weight='0,58'
        RDFNsId1:img='1'>
        <RDFNsId1:title>DAML Project for Stanford
Knowledge Systems Laboratory</RDFNsId1:title>
        <RDFNsId1:cite>DAML-S: DAML-BASED WEB SERVICE
&lt;STRONG&gt;ONTOLOGY&lt;/STRONG&gt; </RDFNsId1:cite>
        <RDFNsId1:weight>1</RDFNsId1:weight>
        <RDFNsId1:cite>&lt;STRONG&gt;DAML&lt;/STRONG&gt; PROJECT FOR STANFORD
KNOWLEDGE SYSTEMS LABORATORY </RDFNsId1:cite>
        </rdf:Description>
        </RDFNsId1:url>

```

```

        <RDFNsId1:url>
        <rdf:Description
rdf:about='http://www.geneontology.org/doc/go.biblio.html'
        RDFNsId1:weight='0,54'
        RDFNsId1:img='5'>
        <RDFNsId1:title>Gene Ontology - A
Bibliography</RDFNsId1:title>
        <RDFNsId1:cite>THE GENE
&lt;STRONG&gt;ONTOLOGY&lt;/STRONG&gt; CONSORTIUM. 2001. CREATING THE
GENE &lt;STRONG&gt;ONTOLOGY&lt;/STRONG&gt;</RDFNsId1:cite>
        </rdf:Description>
        </RDFNsId1:url>
    </rdf:Description>
</RDFNsId1:class>
<RDFNsId1:class>
    <rdf:Description rdf:about='DAML'>
        <rdfs:subClassOf rdf:resource='ontology'>/>
        <RDFNsId1:url>
        <rdf:Description
rdf:about='http://www.lassila.org/'
        RDFNsId1:weight='1'
        RDFNsId1:img='2'>
        <RDFNsId1:title>Ora Lassila</RDFNsId1:title>
        <RDFNsId1:cite>LIST OF MY
&lt;STRONG&gt;PUBLICATIONS&lt;/STRONG&gt; </RDFNsId1:cite>
        </rdf:Description>
        </RDFNsId1:url>
        <RDFNsId1:url
rdf:resource='http://www.ksl.stanford.edu/projects/daml/'>/>
        <RDFNsId1:url>
        <rdf:Description rdf:about='http://www.daml.org/'
        RDFNsId1:title='DAML.org'
        RDFNsId1:weight='1'
        RDFNsId1:img='2'>

<RDFNsId1:cite>&lt;STRONG&gt;DAML&lt;/STRONG&gt;
&lt;STRONG&gt;PUBLICATIONS&lt;/STRONG&gt; </RDFNsId1:cite>
        </rdf:Description>
        </RDFNsId1:url>
        <RDFNsId1:url>
        <rdf:Description
rdf:about='http://www.w3.org/tr/1999/rec-rdf-syntax-19990222/'
        RDFNsId1:weight='1'
        RDFNsId1:img='17'>
        <RDFNsId1:title>Resource Description Framework
(RDF) Model and Syntax Specification</RDFNsId1:title>
        <RDFNsId1:cite>IN THIS EXAMPLE THERE IS NO
STATED RELATIONSHIP BETWEEN THE
&lt;STRONG&gt;PUBLICATIONS&lt;/STRONG&gt;</RDFNsId1:cite>
        </rdf:Description>
        </RDFNsId1:url>
        <RDFNsId1:url>
        <rdf:Description
rdf:about='http://www.w3.org/TR/daml+oil-reference'
        RDFNsId1:weight='1'
        RDFNsId1:cite=''
        RDFNsId1:img='1'>
        <RDFNsId1:title>DAML+OIL (March 2001)
Reference Description</RDFNsId1:title>
        </rdf:Description>
        </RDFNsId1:url>

```

```

        <RDFNSId1:url>
        <rdf:Description
rdf:about='http://www.daml.org/2002/04/why.html'
        RDFNSId1:weight='1'
        RDFNSId1:img='1'>
        <RDFNSId1:title>Why Use DAML?</RDFNSId1:title>
        <RDFNSId1:cite>A SET OF
&lt;STRONG&gt;DAML&lt;/STRONG&gt; STATEMENTS BY ITSELF (AND THE
&lt;STRONG&gt;DAML&lt;/STRONG&gt;</RDFNSId1:cite>
        </rdf:Description>
        </RDFNSId1:url>
        <RDFNSId1:url>
        <rdf:Description
rdf:about='http://www.w3.org/TR/daml+oil-walkthru/'
        RDFNSId1:weight='1'
        RDFNSId1:img='1'>
        <RDFNSId1:title>Annotated DAML+OIL Ontology
Markup</RDFNSId1:title>
        <RDFNSId1:cite>DAML+OIL DIVIDES THE WORLD UP
INTO OBJECTS, WHICH ARE ELEMENTS OF
&lt;STRONG&gt;DAML&lt;/STRONG&gt;</RDFNSId1:cite>
        </rdf:Description>
        </RDFNSId1:url>
        <RDFNSId1:url>
        <rdf:Description rdf:about='http://www.bell-
labs.com/user/pfps/'
        RDFNSId1:weight='1'
        RDFNSId1:img='5'>
        <RDFNSId1:title>Peter F. Patel-
Schneider&apos;s home page</RDFNSId1:title>
        <RDFNSId1:cite>, AS WELL AS THE
&lt;STRONG&gt;THEORY&lt;/STRONG&gt; OF DESCRIPTION LOGICS.
</RDFNSId1:cite>
        </rdf:Description>
        </RDFNSId1:url>
        <RDFNSId1:url>
        <rdf:Description
rdf:about='http://www.w3.org/rdf/'
        RDFNSId1:weight='1'
        RDFNSId1:img='4'>
        <RDFNSId1:title>Resource Description Framework
(RDF) / W3C Semantic Web Activity</RDFNSId1:title>
        <RDFNSId1:cite>RDF MODEL
&lt;STRONG&gt;THEORY&lt;/STRONG&gt;</RDFNSId1:cite>
        </rdf:Description>
        </RDFNSId1:url>
        <RDFNSId1:url>
        <rdf:Description
rdf:about='http://www.xml.com/pub/a/2002/01/30/daml1.html'
        RDFNSId1:weight='1'
        RDFNSId1:cite=''
        RDFNSId1:img='51'>
        <RDFNSId1:title>XML.com: Introduction to DAML:
Part I [Jan. 30, 2002]</RDFNSId1:title>
        </rdf:Description>
        </RDFNSId1:url>
        <RDFNSId1:url>
        <rdf:Description
rdf:about='http://www.ksl.stanford.edu/people/dlm/'
        RDFNSId1:weight='0,85'
        RDFNSId1:img='2'>

```

```

                                <RDFNSId1:title>Deborah L.
McGuinness</RDFNSId1:title>

<RDFNSId1:cite>&lt;STRONG&gt;DAML&lt;/STRONG&gt; WEB SITE
</RDFNSId1:cite>
                                </rdf:Description>
                                </RDFNSId1:url>
                                <RDFNSId1:url
rdf:resource='http://www.ksl.stanford.edu/projects/rkf/'/>
                                <RDFNSId1:url>
                                <rdf:Description
rdf:about='http://www.w3.org/xml/'
                                RDFNSId1:weight='0,71'
                                RDFNSId1:img='7'>
                                <RDFNSId1:title>Extensible Markup Language
(XML)</RDFNSId1:title>
                                <RDFNSId1:cite>TIMELINE: EVENTS AND
&lt;STRONG&gt;PUBLICATIONS&lt;/STRONG&gt; </RDFNSId1:cite>
                                </rdf:Description>
                                </RDFNSId1:url>
                                <RDFNSId1:url>
                                <rdf:Description rdf:about='http://www-
db.stanford.edu/~stefan/'
                                RDFNSId1:weight='0,68'
                                RDFNSId1:img='3'>
                                <RDFNSId1:title>Stefan Decker</RDFNSId1:title>
                                <RDFNSId1:cite>DARPA
&lt;STRONG&gt;DAML&lt;/STRONG&gt; PROGRAM </RDFNSId1:cite>
                                </rdf:Description>
                                </RDFNSId1:url>
                                <RDFNSId1:url>
                                <rdf:Description
rdf:about='http://rdfweb.org/people/danbri/'
                                RDFNSId1:weight='0,6'
                                RDFNSId1:img='2'>
                                <RDFNSId1:title>Dan Brickley</RDFNSId1:title>

<RDFNSId1:cite>&lt;STRONG&gt;DAML&lt;/STRONG&gt; </RDFNSId1:cite>
                                </rdf:Description>
                                </RDFNSId1:url>
                                <RDFNSId1:url>
                                <rdf:Description
rdf:about='http://www.lcs.mit.edu/'
                                RDFNSId1:weight='0,6'
                                RDFNSId1:img='26'>
                                <RDFNSId1:title>Welcome to the MIT Laboratory
for Computer Science</RDFNSId1:title>

<RDFNSId1:cite>&lt;STRONG&gt;PUBLICATIONS&lt;/STRONG&gt;
</RDFNSId1:cite>
                                </rdf:Description>
                                </RDFNSId1:url>
                                </rdf:Description>
                                </RDFNSId1:class>
                                <RDFNSId1:class>
                                <rdf:Description rdf:about='RDF'>
                                <rdfs:subClassOf rdf:resource='ontology'/'>
                                <RDFNSId1:url>
                                <rdf:Description
rdf:about='http://www.w3.org/tr/rec-rdf-syntax/'
                                RDFNSId1:weight='1'

```

```

        RDFNsId1:img='17'>
        <RDFNsId1:title>Resource Description Framework
(RDF) Model and Syntax Specification</RDFNsId1:title>
        <RDFNsId1:cite>THIS DOCUMENT INTRODUCES A
MODEL FOR REPRESENTING
&lt;STRONG&gt;RDF&lt;/STRONG&gt;</RDFNsId1:cite>
        </rdf:Description>
        </RDFNsId1:url>
        <RDFNsId1:url>
        <rdf:Description
rdf:about='http://www.w3.org/TR/REC-rdf-syntax/'
        RDFNsId1:weight='1'
        RDFNsId1:img='17'>
        <RDFNsId1:title>Resource Description Framework
(RDF) Model and Syntax Specification</RDFNsId1:title>
        <RDFNsId1:cite>THIS DOCUMENT INTRODUCES A
MODEL FOR REPRESENTING
&lt;STRONG&gt;RDF&lt;/STRONG&gt;</RDFNsId1:cite>
        </rdf:Description>
        </RDFNsId1:url>
        <RDFNsId1:url>
        <rdf:Description rdf:about='http://nestroy.wi-
inf.uni-essen.de/rdf/sum_rdf_api/'
        RDFNsId1:weight='1'
        RDFNsId1:img='4'>
        <RDFNsId1:title>An overwiev about recent
developments in Application Programming Interfaces for
RDF</RDFNsId1:title>
        <RDFNsId1:cite>CHAPTER 2 IS DEDICATED TO
PREVIOUS APPROACHES TO AN
&lt;STRONG&gt;RDF&lt;/STRONG&gt;</RDFNsId1:cite>
        </rdf:Description>
        </RDFNsId1:url>
        <RDFNsId1:url>
        <rdf:Description
rdf:about='http://www.xml.com/pub/a/2001/01/24/rdf.html'
        RDFNsId1:weight='1'
        RDFNsId1:img='49'>
        <RDFNsId1:title>XML.com: What is RDF? [Jan.
24, 2001]</RDFNsId1:title>
        <RDFNsId1:cite>DAN BRICKLEY, CHAIR OF THE
W3C&apos;S &lt;STRONG&gt;RDF&lt;/STRONG&gt;</RDFNsId1:cite>
        </rdf:Description>
        </RDFNsId1:url>
        <RDFNsId1:url>
        <rdf:Description
rdf:about='http://www.ilrt.bristol.ac.uk/discovery/rdf/resources/'
        RDFNsId1:weight='1'
        RDFNsId1:img='2'>
        <RDFNsId1:title>Dave Beckett&apos;s Resource
Description Framework (RDF) Resource Guide</RDFNsId1:title>
        <RDFNsId1:cite>- AN
&lt;STRONG&gt;RDF&lt;/STRONG&gt; PARSER IN JAVA OFFERING FULL-FLEDGED
VALIDATION OF &lt;STRONG&gt;RDF&lt;/STRONG&gt;</RDFNsId1:cite>
        </rdf:Description>
        </RDFNsId1:url>
        <RDFNsId1:url>
        <rdf:Description
rdf:about='http://www.w3.org/2000/03/rdf-tracking/'
        RDFNsId1:weight='1'
        RDFNsId1:img='1'>

```

```

        <RDFNSId1:title>RDF Issue
Tracking</RDFNSId1:title>
        <RDFNSId1:cite>EXAMPLES IN THE
&lt;STRONG&gt;RDF&lt;/STRONG&gt;</RDFNSId1:cite>
        </rdf:Description>
        </RDFNSId1:url>
        <RDFNSId1:url>
        <rdf:Description
rdf:about='http://www.mozilla.org/rdf/doc/'
        RDFNSId1:weight='1'
        RDFNSId1:img='1'>
        <RDFNSId1:title>resource description
framework</RDFNSId1:title>
        <RDFNSId1:cite>FOR EXAMPLE, THERE IS AS YET NO
AGREED &lt;STRONG&gt;RDF&lt;/STRONG&gt;</RDFNSId1:cite>
        </rdf:Description>
        </RDFNSId1:url>
        <RDFNSId1:url>
        <rdf:Description
rdf:about='http://www.ilrt.bris.ac.uk/discovery/rdf/resources/'
        RDFNSId1:weight='1'
        RDFNSId1:img='2'>
        <RDFNSId1:title>Dave Beckett's Resource
Description Framework (RDF) Resource Guide</RDFNSId1:title>
        <RDFNSId1:cite>- AN
&lt;STRONG&gt;RDF&lt;/STRONG&gt; PARSER IN JAVA OFFERING FULL-FLEDGED
VALIDATION OF &lt;STRONG&gt;RDF&lt;/STRONG&gt;</RDFNSId1:cite>
        </rdf:Description>
        </RDFNSId1:url>
        <RDFNSId1:url>
        <rdf:Description
rdf:about='http://www.w3.org/RDF/'
        RDFNSId1:weight='1'
        RDFNSId1:img='4'>
        <RDFNSId1:title>Resource Description Framework
(RDF) / W3C Semantic Web Activity</RDFNSId1:title>

        <RDFNSId1:cite>&lt;STRONG&gt;RDF&lt;/STRONG&gt; VOCABULARY DESCRIPTION
LANGUAGE 1.0: &lt;STRONG&gt;RDF&lt;/STRONG&gt; SCHEMA </RDFNSId1:cite>
        </rdf:Description>
        </RDFNSId1:url>
        <RDFNSId1:url>
        <rdf:Description
rdf:about='http://rdfig.xmlhack.com/index.html'
        RDFNSId1:weight='0,77'
        RDFNSId1:img='0'>
        <RDFNSId1:title>RDF Interest Group IRC
Scratchpad, last cranked at 2002-09-24 20:57</RDFNSId1:title>
        <RDFNSId1:cite>IDEAS ON THE USE OF
&lt;STRONG&gt;RDF&lt;/STRONG&gt; QUERY FOR METADATA ABOUT
&lt;STRONG&gt;RDF&lt;/STRONG&gt; VOCABS </RDFNSId1:cite>
        </rdf:Description>
        </RDFNSId1:url>
        </rdf:Description>
        </RDFNSId1:class>
        </rdf:Description>
</rdf:RDF>

```

Juego de pruebas Distribuido

La realización de este juego de pruebas se ha realizado sólo para el caso de tres agentes. La distribución de los agentes ha sido en dos máquinas. En la primera, se han colocado dos $A_{internet}$ y el A_{coord} . En la segunda, únicamente un sólo $A_{internet}$. Este agente solitario se ha ocupado exclusivamente de la clase **DAML**.

Las sentencias que se han usado para arrancar los agentes han sido:

**Java jade.Boot -gui a1:Sliver.InternetAgent a2:Sliver.InternetAgent
cA:Sliver.CoordinatorAgent**

para la primera computadora y

Java jade.Boot -host alegria -container a3:Sliver.InternetAgent

para la segunda.

Podemos observar gráficamente este hecho en el **RMA** de la plataforma:

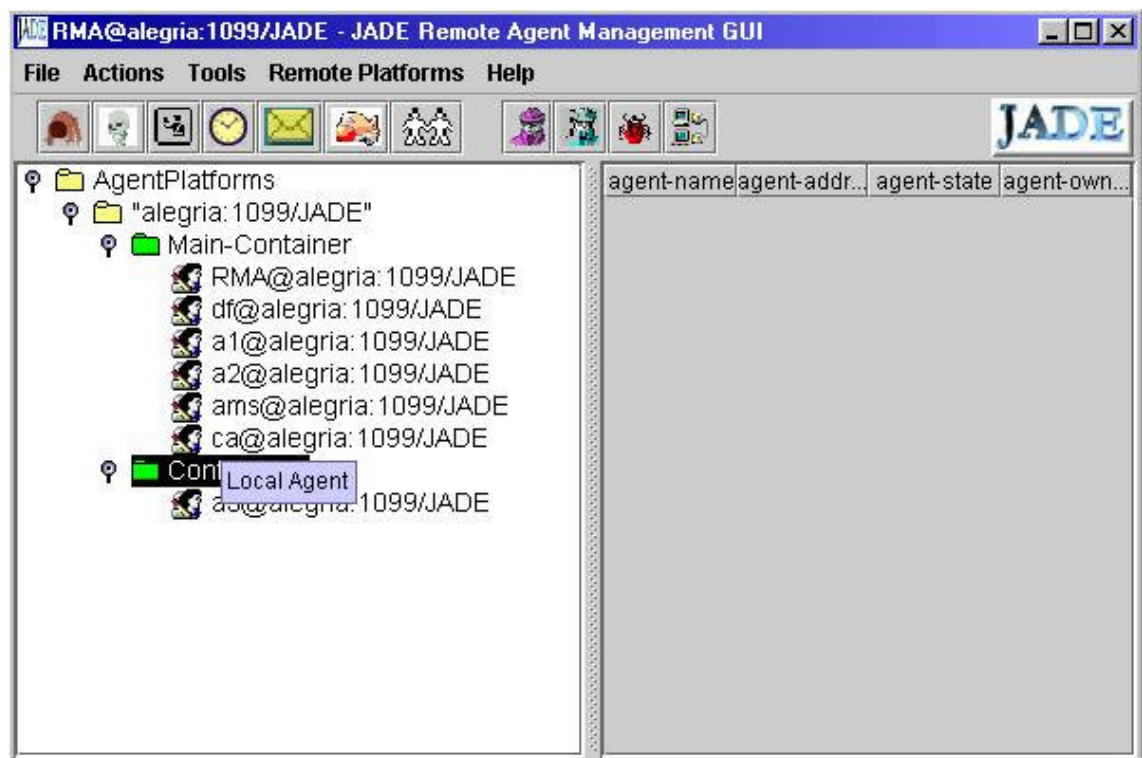
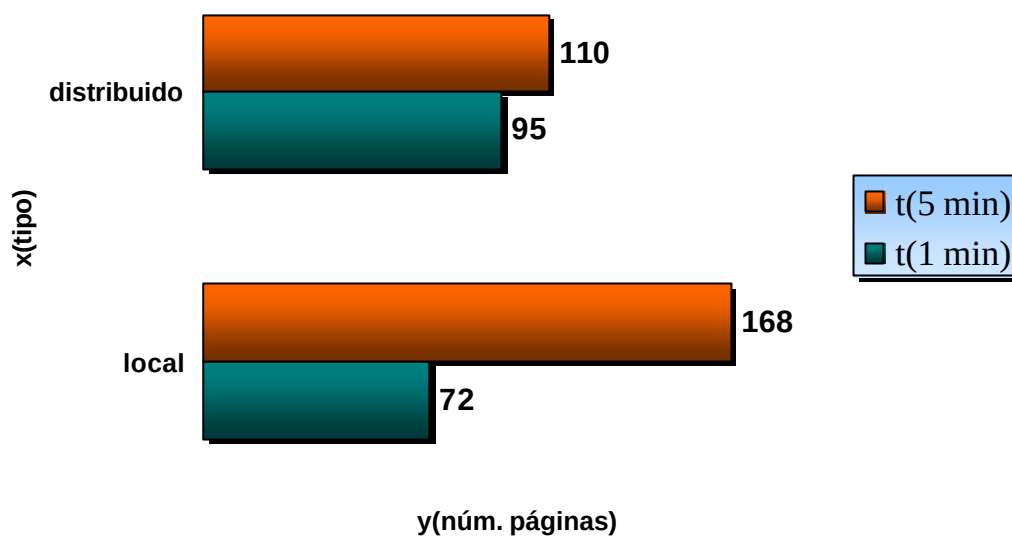


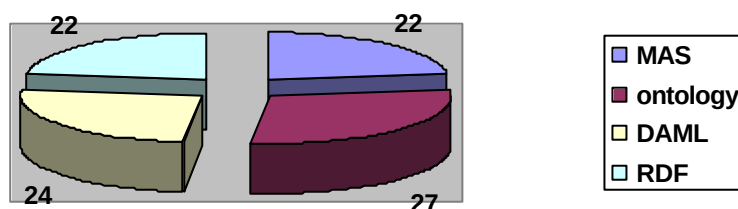
Figura 13. Muestra el estado de RMA en el juego de pruebas distribuido.

A continuación, mostramos los resultados de la prueba distribuida en forma de gráficas. En la primera se comparan el juego de pruebas local con el juego de pruebas distribuido para el caso de 3 agentes. También en esta ocasión, se han tomado los dos tiempos. En las restantes se ven los resultados distribuidos por clases.

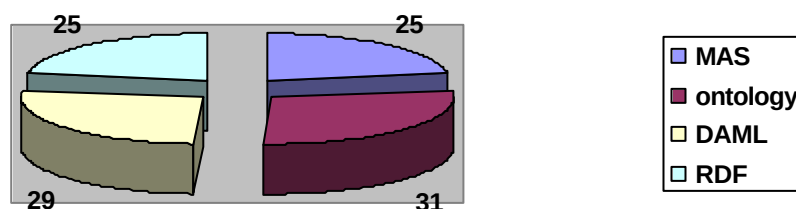


A continuación mostramos las gráficas de las distribuciones por clases para el caso del juego de pruebas distribuido. Para el caso del juego de pruebas local para tres agentes, las gráficas ya se han mostrado con anterioridad.

Distribución de clases para 3 agentes en 1 minuto



Distribución de clases para 3 agentes en 5 minutos



Como podemos observar en las anteriores gráficas, el hecho de distribuir los agentes en dos máquinas tampoco ha supuesto un cambio substancial. Se puede

observar que la única diferencia así apreciable ha sido que las ocurrencias para la clase ***DAML*** han pasado de 21 a 29. Esto es porque el agente responsable de esta clase ha sido aislado en una de las máquinas.

8. Conclusiones y Trabajo Futuro

La realización de este proyecto ha supuesto un gran esfuerzo personal. Todas las técnicas utilizadas en el desarrollo eran casi totalmente desconocidas. Por tanto, el desarrollo de este proyecto ha supuesto un gran aprendizaje, especialmente en el campo de los sistemas MultiAgentes y de los lenguajes de metadatos tales como RDF. Además, ha permitido obtener una visión de mayor nivel del estado de Internet y de cuáles son las propuestas que intentan solventar sus deficiencias.

La capacidad de los sistemas MultiAgentes de trabajar de forma distribuida ha permitido dotar a este proyecto de cierto *sentido*. De esta forma, el buscador puede distribuirse en varias máquinas y obtener mayores resultados. Además, el uso de ontologías en RDF como terminología de búsqueda, ha permitido proporcionarle mayor capacidad de selección.

Todo esto ha hecho que este proyecto constituya una buena base para desarrollar en un futuro buscadores inteligentes que proyectos como h-techsight puedan usar.

Sin embargo, algunas cuestiones no han quedado resueltas y en un futuro se podrían intentar solucionar para mejorar este buscador:

Por ejemplo, se podría hacer que los $A_{internet}$ se comunicarán los unos con los otros y fueran ellos quienes fueran eliminando redundancias. También, se podría hacer que trabajaran de forma jerarquizada, de forma que cada $A_{internet}$ padre fuera responsable de filtrar la información de los $A_{internet}$ hijos etc. Todo esto constituirían modificaciones que aumentarían la eficiencia. También, se podría usar algún agente que gestionará el conocimiento aprendido en los links de los $A_{internet}$ y que descubriera nuevos conceptos a partir de ello, mandando el conocimiento nuevo a los $A_{internet}$. Estas modificaciones permitirían en un futuro generar herramientas que permitieran extraer conceptos e información de Internet de manera inteligente.

En resumen, este proyecto ha abierto las puertas a un mundo aún por descubrir y del cual nos podemos beneficiar considerablemente.

10. Referencias

- [1] Especificación oficial XML v1.0 (<http://www.w3.org/TR/REC-xml>)
- [2] The World Wide Web Consortium (<http://www.w3c.org>)
- [3] David Isern, Projecte Final de Carrera, URV, “*Avaluació d’entorns de desenvolupament de Sistemes Multiagent*”, 1998-1999
- [4] Galan, Alan, “*Jafmas complete thesis*”, Department of Electrical and Computer Engineering and Computer Science, 1998.
(<http://www.ececs.uc.edu/~abaker/JAFMAS>)
- [5] Nwana, H (BT). y Wooldridge, M. (QMW), “*Software Agent Technologies*”, BT Labs. (http://www.labs.bt.com/agents/publish/papers/sat_report.htm)
- [6] Golobardes, E., Pérez, J., Porta, J.M., “*Tutorial sobre agents*”, Seminario de Inteligencia Artificial (UPC), 1996.
- [7] Foundation for Intelligent Physical Agents, (<http://www.fipa.org>)
- [8] Foundation for Intelligent Physical Agents, “*FIPA 99 Specification*”, FIPA, Ginebra, Suiza, 1999.
- [9] Odell, P.; Dyke, H. and Bahuer, B. Extending UML for Agents. AIOS WorkShop in AAI’0 02000
- [10] Simple API for XML – SAX (<http://www.developerlife.com>)
- [11] Estándar W3C DOM (<http://www.w3.org/DOM>)
- [12] Java Networking Programming; Eliotte Rusty Harold. ISBN 1 –5692-870-9.
- [13] “JADE programmer’s guide”, 2002
- [14] JENA RDF API
(http://www.hpl.hp.com/semweb/iswc2002/JenaTutorial.Alpha/RDF_API/tutorial.html)
- [15] GOOGLE API (<http://www.google.com/apis/>)
- [16] Java HTML JTIDY API(<http://tidy.sourceforge.net>)