UNIVERSIDAD POLITÉCNICA DE MADRID

FACULTAD DE INFORMÁTICA

TRABAJO FIN DE CARRERA

DESARROLLO DE UN SISTEMA DE MEDIACIÓN SEMÁNTICA, BASADO EN ONTOLOGÍAS, DE BASES DE DATOS HETEROGÉNEAS SOBRE TECNOLOGÍAS GRID

AUTORA: Ana Jiménez Castellanos

TUTOR: Víctor Manuel Maojo García

COTUTOR: Luis Martín Martín

Febrero 2009

A mi familia.

Muchas gracias por ser como sois.

AGRADECIMIENTOS

Me gustaría agradecer a todas aquellas personas que de una forma u otra han estado a mi lado a lo largo de esta carrera.

En primer lugar quisiera dar las gracias a mi familia, especialmente a mis padres y hermano, Juan, María del Carmen, y Juan, por apoyarme durante toda la vida, en los buenos y malos momentos, y por haberme enseñado a aprender. También me gustaría dar las gracias a mis abuelos, por ser un referente desde hace muchos años.

A Guillermo, por entenderme y ayudarme desde hace tanto tiempo. Por ser mi compañero y por su impagable dominio del Word. Te quiero.

A mi tutor, el catedrático Víctor Maojo, por confiar en mí y darme la oportunidad de trabajar en el Grupo de Informática Biomédica.

A las generaciones de compañeros y amigos que he visto pasar por el laboratorio en todo este tiempo, por enseñarme a trabajar en equipo; en especial a Alberto y Luis por su paciencia y dedicación. Gracias a todos por vuestros consejos, tanto profesionales como personales.

A la Asociación de teatro universitario Histrión, por hacerme disfrutar del teatro y del trabajo en equipo.

A TEDECO, por darme la oportunidad de conocer Burundi, por enseñarme que no sólo existen salidas hacia el primer mundo.

CONTENIDOS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

1.1 PLANTEAMIENTO DEL PROBLEMA

En los últimos años, la cantidad de información producida y almacenada en Bases de Datos ha crecido enormemente. Concretamente, en el campo de la biomedicina, la producción de datos ha sido incesante. Lógicamente, este volumen ingente de datos ha sido compilado en multitud de Bases de Datos y sistemas de información dispares usando lenguajes distintos, por lo que gestionar toda esa información proveniente de diversas fuentes heterogéneas es uno de los grandes retos de la informática actual. Por ejemplo, el proyecto “Genoma Humano” ha generado una gran cantidad de información que ha sido recogida en distintas fuentes de datos. Para controlar y manejar toda esa cantidad de información, en los últimos años ha surgido una nueva área dentro de la informática: la integración semántica de Bases de Datos.

La información semántica asociada al área de la biomedicina es bastante amplia, y por tanto, la resolución de heterogeneidades semánticas (entre otras) es básica para la integración de información cuando se manejan datos complejos. Por esta razón, los sistemas de integración, cada vez más a menudo, emplean ontologías para dotar a las herramientas del poder semántico que éstas ofrecen. Algunos sistemas actuales usan las ontologías como soporte formal a la información que define las transformaciones sobre los datos; otros las emplean como almacenes de información para las fuentes de datos.

Este proyecto se enmarca dentro de ACGT (Advancing Clinico-Genomic Trials on Cancer). ACGT es un proyecto integrado financiado por la Comisión Europea (FP6-2005-IST-026996) en el que participan 25 organizaciones, europeas y una japonesa. Está dedicado al desarrollo de tecnologías para dar soporte a ensayos clínicos post-genómicos [1]. El objetivo principal del proyecto ACGT es construir una plataforma que ayude a los investigadores biomédicos en sus investigaciones contra el cáncer, en concreto el llamado tumor de Wilm y el cáncer de mama. El proyecto ofrecerá una serie de servicios para: el desarrollo de ensayos clínicos, herramientas analíticas,

herramientas de simulación de crecimiento de tumores y servicios de consulta a Bases de Datos heterogéneas.

ACGT se basa en tres ideas fundamentales:

- Grid: se desea desarrollar un middleware de servicios Grid que permitan la interoperabilidad entre recursos y servicios biomédicos.
- Integración: de datos heterogéneos y distribuidos.
- Descubrimiento de conocimiento entre los datos integrados.

El entorno de ACGT promueve la creación de unas organizaciones europeas para la investigación clínica. Estas organizaciones tendrían la forma de organizaciones virtuales para la investigación en cáncer.

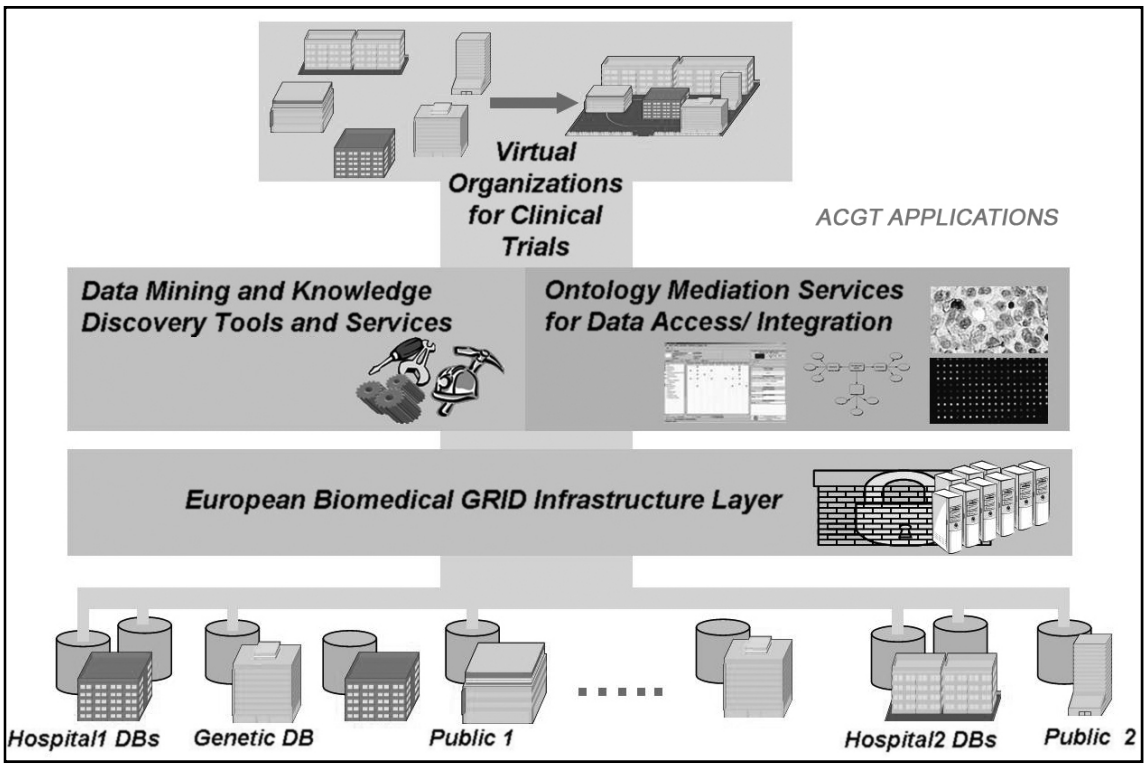

Fig 1-1 Arquitectura de ACGT [1]

Una organización virtual es creada en el entorno de ACGT mediante la integración de datos distribuidos y heterogéneos. Dicha integración se realiza por medio de

servicios de mediación basados en ontologías, además de a través de la inclusión de unas herramientas y servicios de minería de datos y descubrimiento de conocimiento, todo ello sobre una capa Grid, tal y como se muestra en la figura Fig 1-1.

Debido al enorme crecimiento en tamaño y número de Bases de Batos biomédicas durante los últimos años, la correcta integración semántica se ha convertido en un rasgo crítico en el desarrollo del proyecto ACGT. La mediación semántica permitirá a los usuarios finales (clínicos) aumentar la productividad de los ensayos clínicos realizados, liberándoles de los problemas que los datos heterogéneos y distribuidos presentan. También los experimentos que realicen serán más fructíferos pues podrán acceder a más información de forma más sencilla. Además, la integración de fuentes de datos heterogéneas y distribuidas puede conducir al descubrimiento de nuevas relaciones semánticas entre las distintas fuentes.

1.2 OBJETIVOS

En este proyecto se busca desarrollar un middleware (una interfaz común) que responda a consultas de los clientes sobre ensayos clínicos, a partir de un conjunto integrado de bases de datos. Dicho conjunto está compuesto por fuentes de datos distintas y dispersas.

El objetivo principal es la integración de fuentes de datos heterogéneas, ofreciendo una serie de servicios de consulta contra las mismas y de forma que se oculte al usuario la complejidad del proceso de traducción de consultas y la integración de los resultados. Se pretende también resolver las inconsistencias tanto a nivel de esquema como a nivel de instancia.

Como objetivos secundarios, añadidos a esta capa que se encargará de mediar semánticamente entre los usuarios y las fuentes de datos heterogéneas, se muestran los siguientes:

- Generar un modelo semántico de metadatos que capture la representación de los esquemas locales y del esquema global.
- Actualizar automáticamente los esquemas locales de cada fuente de datos.

- Generar metadatos sobre los datos de las consultas realizadas.
- Colocar los resultados de las consultas en una plataforma integrada.
- Conseguir un conjunto de mediadores operacionales (*wrappers*) que envíen las subconsultas a las fuentes de datos.
- Generar un servicio de consulta a un servidor para conocer las fuentes de datos disponibles y sus esquemas.

1.3 SOLUCIÓN PROPUESTA

Se pretende realizar una capa de mediación semántica dentro del ambiente ACGT con la estructura mostrada en la figura Fig 1-2. Esta capa está basada en el enfoque híbrido de traducción de preguntas y contiene un conjunto de aplicaciones o servicios:

ACGT-MO

La ACGT-MO (*Master Ontology* u Ontología Maestra) es una ontología que recoge la información sobre cáncer que se maneja en el entorno de ACGT. Consiste en una base conceptual para un repositorio de conocimiento estructurado sobre cáncer. Es muy útil por servir para realizar la mediación semántica entre usuarios y fuentes de datos de dos formas:

- Se usa en el proceso de integración para crear *mappings* (conjunto de correspondencias entre términos físicos y términos de la Ontología Maestra)
- El usuario realiza las consultas en términos de la Ontología Maestra.

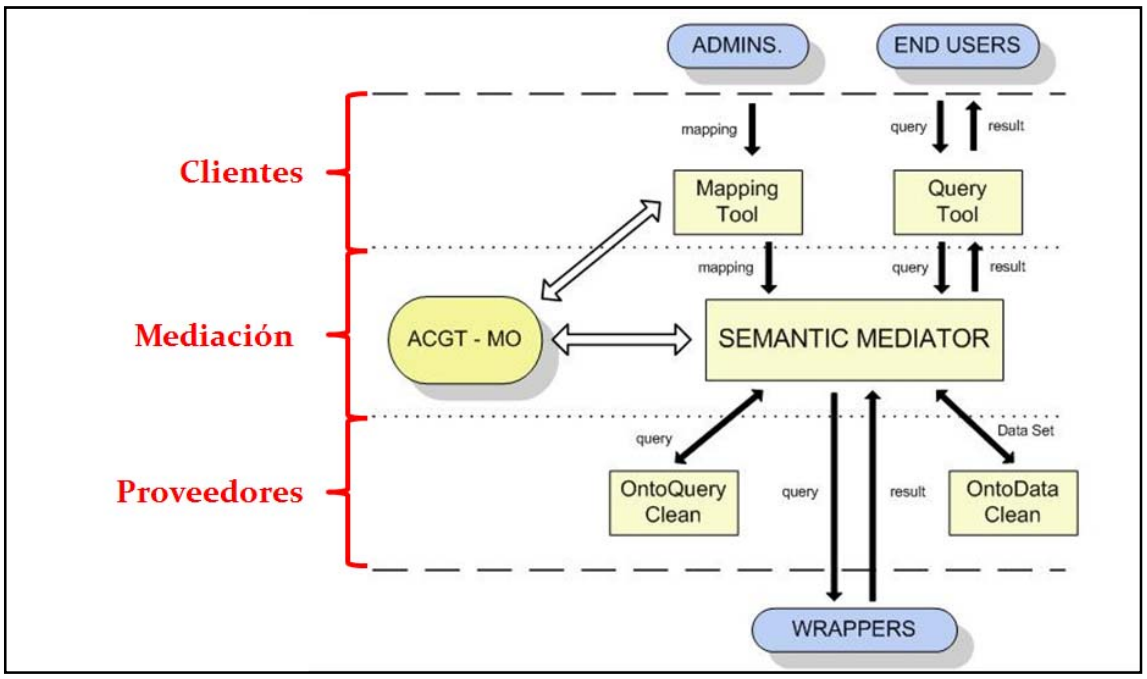

Fig 1-2 Capa de Mediación Semántica de ACGT

Semantic Mediator

El *Semantic Mediator* (o mediador semántico) es el "corazón" del proceso de integración. Es un módulo software que resuelve las heterogeneidades semánticas existentes entre las fuentes de datos distribuidas.

Cuando recibe una consulta se encarga de:

- Dividirla entre las distintas fuentes de datos a las que vaya dirigida la consulta.
- Lanzar esas subconsultas a los *wrappers* (envoltorios de fuentes de datos) que sean necesarios.
- Recoger e integrar los distintos resultados para mostrárselos de forma homogénea al usuario.

Utiliza la Ontología Maestra y los *mappings* para resolver las heterogeneidades semánticas.

Mapping Tool

Es una herramienta basada en Web que se utiliza para el desarrollo colaborativo de *mappings*.

Esta herramienta se encarga de:

- Gestión de usuarios de forma que diferentes usuarios puedan utilizar la herramienta simultáneamente para construir los *mappings*.
- Gestión de mappings permitiendo la construcción de correspondencias semánticas que resuelvan la heterogeneidad a nivel de esquema.
- Control de coherencia entre las colaboraciones de distintos expertos.

Query Tool

Es una herramienta basada en Web que se utiliza para realizar consultas contra el mediador semántico. Tiene una serie de características:

- Se encarga de la gestión de usuarios y de los distintos roles de uso que pueden tener los mismos.
- Realiza una gestión de consultas pues, además de lanzar las consultas y recoger los resultados, permite el almacenamiento de las consultas para modificarlas o usarlas posteriormente.
- Tiene una comunicación con el mediador semántico en tiempo real para permitir al usuario construir consultas correctas y sobre elementos existentes.

OntoDataClean

Es un sistema basado en ontologías, orientado al preprocesamiento automático de datos para resolver heterogeneidades a nivel de instancia. Resuelve y elimina distintas inconsistencias en los datos como pueden ser: transformaciones de escala o rango, eliminación de valores perdidos, cambio de formato, etc.

OntoQueryClean

Es un sistema basado en ontologías cuyo funcionamiento es muy similar al del sistema OntoDataClean; se utiliza también para el preprocesamiento, pero en este caso de las consultas que puedan llegar al mediador semántico.

En este Trabajo de Fin de Carrera se han desarrollado una serie de servicio basados en la tecnología proporcionada por OGSA-DAI y Globus Toolkit, tecnologías que proporcionan la capa de Grid que necesitan los servicios. Todos los servicios han sido desarrollados en el lenguaje de programación Java.

Como se ha comentado, el enfoque utilizado para la integración de fuentes de datos heterogéneas ha sido el híbrido de traducción de preguntas, por trabajar dentro del entorno de ACGT.

En las siguientes secciones se muestra de forma más detallada cada uno de los servicios que se han implementado.

1.3.1 Mediador Semántico

Es un servicio Grid que resuelve consultas generadas por los usuarios. Dispone de un conjunto de *mappings* para resolver la integración, uno por cada Base de Datos que deba ser incluida en el proceso de integración (ver figura Fig 1-3)

El funcionamiento del mediador semántico es el siguiente:

- Se recibe la consulta en términos conceptuales (en términos de la ontología maestra de ACGT).
- Se comprueba qué atributos, relaciones y restricciones de la consulta inicial están incluidos en cada uno de los *mappings* del sistema, dividiéndose la consulta en las diferentes subconsultas que atañen a cada *mapping*.
- Se traduce la subconsulta a términos físicos haciendo uso de las traducciones contenidas en los respectivos *mappings*.
- Se lanza la subconsulta a cada *wrapper* (siempre que exista subconsulta para ese *wrapper*) y se recogen los diferentes resultados de cada uno de ellos.
- Se integran los resultados:

- Se vuelven a traducir los atributos, esta vez de términos físicos a conceptuales.
- Se "reúnen" todos los resultados en un repositorio virtual que será destruido cuando se haya respondido a la consulta.

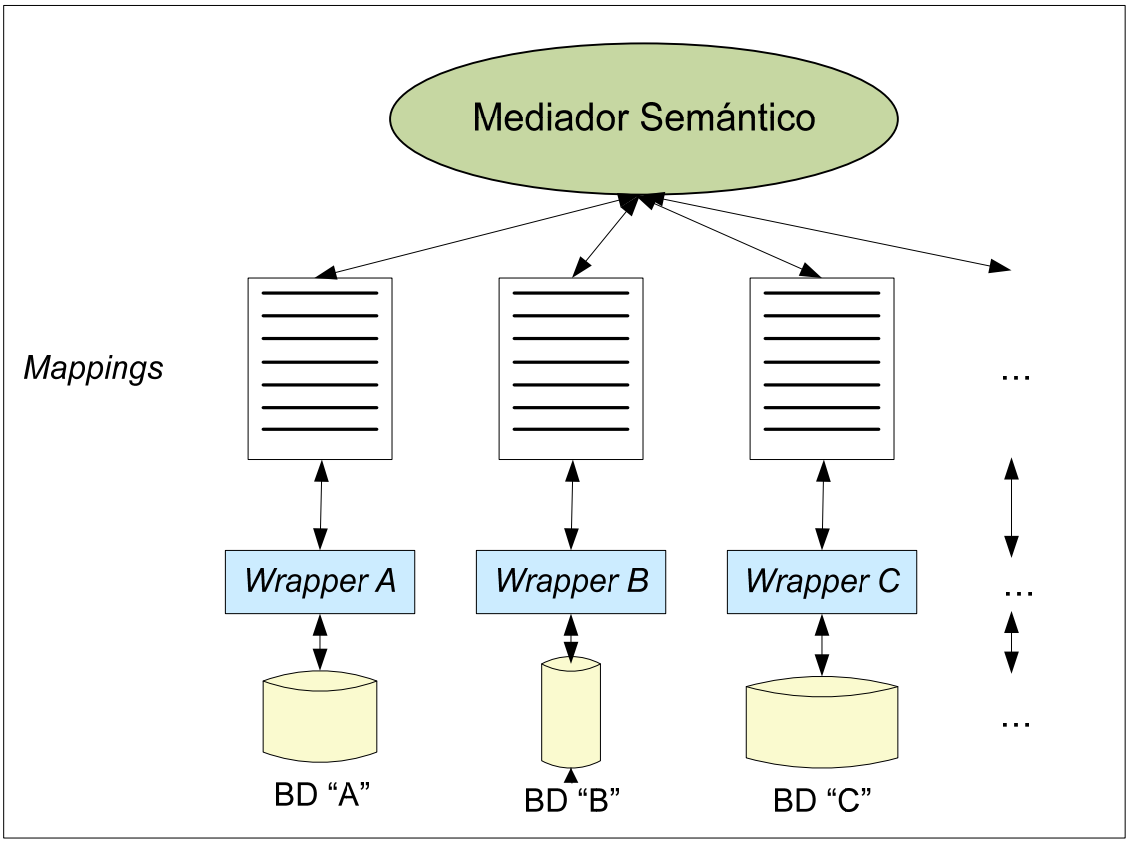

Fig 1-3 Esquema del mediador semántico

- Se imponen las restricciones (si las hubiera) sobre ese repositorio que incluye todos los datos que tiene el sistema.
- Se devuelven los resultados

También se llama mediador semántico al conjunto de servicios formados por el comentado hasta el momento y el resto de servicios que ofrece la solución propuesta a las herramientas de gestión de *mappings* y de gestión de consultas.

1.3.2 Servicios útiles para la MappingTool

La MappingTool o “herramienta de gestión de *mappings*” necesita disponer de dos servicios para su correcto funcionamiento:

- La MapingTool necesita ser capaz de avisar al mediador semántico cuando cierto *mapping* ya existente haya sido modificado o cuando un experto haya creado uno nuevo. La solución propuesta incluye un servicio Grid para dar respuesta a esta necesidad. La MappingTool envía el *mapping* creado al mediador semántico y éste actualiza su sistema dependiendo de si se trata de de una modificación de un *mapping* ya existente o de la creación de uno nuevo.
- Por otra parte, esta herramienta debe poder conocer los *wrappers* que hay disponibles en el mediador semántico para poder crear conjuntos de correspondencias entre los términos físicos que manejan dichos *wrappers* y la ontología maestra. Por ello existe otro servicio Grid que lista los *wrappers* de los que dispone el mediador semántico para resolver las consultas.

1.3.3 Servicios útiles para la QueryTool

Por una parte la QueryTool o “herramienta de gestión de consultas”, se encarga de lanzar consultas contra el mediador semántico y de presentar al usuario los resultados a las mismas.

Por otra parte también es una interfaz que utilizan los usuarios para construir las consultas. Para ello necesita conocer sobre qué datos puede realizar una pregunta al mediador semántico. La solución propuesta incluye por tanto otro servicio Grid que comunica a la herramienta de gestión de consultas las descripciones de los datos que integra.

2 ESTADO DE LA CUESTIÓN

En este capítulo se introduce la base científica sobre la que se fundamentan las diferentes soluciones adoptadas en este trabajo de fin de carrera. En primer lugar se describen los distintos enfoques existentes sobre integración de Bases de Datos y mediación semántica, además de las herramientas más relevantes para llevar a cabo la integración de fuentes heterogéneas de datos. A continuación se realizará una breve descripción de los lenguajes, tecnologías y estándares utilizados en el desarrollo de este proyecto.

2.1 MEDIACIÓN SEMÁNTICA E INTEGRACIÓN DE BASES DE DATOS

La integración de Bases de Datos es el área de la informática dedicada al intercambio y recolección de información; normalmente desde fuentes de datos heterogéneas y dispares.

La integración tiene como objetivo proveer el acceso uniforme a múltiples fuentes de datos autónomas y heterogéneas. Es decir, intenta desarrollar una tecnología que, de forma transparente a los usuarios, trate de consultar y/o actualizar distintas fuentes de datos que sigan diferentes modelos de datos y esquemas, de forma que no se afecte a la conducta de dichas fuentes de datos.

En los últimos años, las Bases de Datos biomédicas se han incrementado tanto en número como en tamaño; también se ha incrementado el número de localizaciones donde se pueden encontrar dichas fuentes de datos. En este entorno científico a nivel mundial, en el que los datos pueden estar distribuidos entre distintos entornos, se introduce una cuestión informática que debe ser resuelta: la heterogeneidad.

El uso de Bases de Datos distintas (que tienden a almacenar los datos usando plataformas y software distintos, con formatos a menudo incompatibles, o con la necesidad de un sistema de acceso o un lenguaje específico) provoca que la integración de los datos sea una tarea tediosa y que consume mucho tiempo. La mediación

semántica está dirigida a este problema. Su objetivo final es el de ofrecer una integración de Bases de Datos distribuidas y heterogéneas, permitiendo a usuarios finales consultar varias Bases de Datos, ofreciendo un sistema de acceso simple y homogéneo.

La integración de Bases de Datos biomédicas tiene una importancia crítica debido a la interconexión conceptual que existe entre los distintos campos de la investigación en biomedicina. Este hecho queda patente en el crecimiento de número de proyectos de integración de Bases de Datos en biomedicina que se ha experimentado en los últimos años.

Resumiendo, existen tres problemas fundamentales en la integración de fuentes de datos:

- Las diferentes fuentes de información son, normalmente, independientes, autónomas y tienen esquemas con estructuras totalmente diferentes, al igual que ocurre con el formato de los datos que almacenan.
- Cada sistema de información tiene su propio mecanismo de consulta. Por ello, una de las tareas de la integración será la construcción de un mecanismo de consulta uniforme sobre aquellos mecanismos existentes para cada una de las fuentes individuales.
- La “incompatibilidad semántica” entre las distintas fuentes de datos; es decir, la heterogeneidad semántica.

Los Sistemas Gestores de Bases de Datos (SGBD) son hoy en día las herramientas más eficientes en el manejo de grandes cantidades de información. Los SGBD permiten la recuperación de los datos almacenados a través de consultas de filtrado así como un rendimiento optimizado para su gestión. Estas soluciones de almacenamientos y tratamiento de datos, a su vez han creado nuevos retos, como el acceso integrado a datos almacenados en distintos sistemas de bases de datos heterogéneos, localizados normalmente en lugares remotos. En una primera clasificación, se puede diferenciar los tipos de integración en tres[2]:

- Integración vertical: la unificación se realiza entre información semánticamente similar y proveniente de distintas fuentes (por ejemplo, un “repositorio virtual”

que dé acceso a los datos sobre mamografías de Estados Unidos recogidos en diferentes Bases de Datos[3]).

- Integración Horizontal: se realiza una composición de información semánticamente complementaria desde distintas fuentes heterogéneas (por ejemplo, un sistema que responda a consultas complejas sobre fuentes de información clínica, genómica y proteómica [4]).
- Integración para la portabilidad de aplicaciones: se realiza una estandarización del acceso a la información semánticamente similar en fuentes de datos dispares (por ejemplo, una interfaz universal para aplicaciones de decisiones de soporte que permita compartir la información entre distintas instituciones sin cambiar la implementación de cada una de las fuentes [5]).

Sin embargo, en los SGBD pueden existir conflictos de representación. De esta forma, las dificultades al realizar la integración de bases de datos, no sólo se dan desde un punto de vista técnico al utilizar distintos SGBD en distintas localizaciones. El mayor problema se da porque la misma información puede estar representada de distinta forma, incluso usando el mismo SGBD. A menudo, las fuentes de datos no tienen un modelo de datos o un esquema bien definido, por lo que la integración de fuentes de información puede llegar a ser una tarea complicada. Además existen inconsistencias semánticas y sintácticas que deben ser resueltas para que el usuario final tenga la percepción de que está trabajando en una base de datos local con toda la información de las fuentes de datos integradas.

Desde este punto de vista de los conflictos a resolver, existen dos tipos de integración:

- A nivel de esquema: trata sobre la unificación de esquemas diferentes para obtener uno común. Por ejemplo, podría darse que campos similares tuvieran nombres distintos en dos bases de datos, o lo que en una base de datos fuera una relación, en otra fuera un atributo.
- A nivel de instancia: trata con la integración del contenido de dichos esquemas. Podría suceder que en dos columnas de dos bases de datos tuviéramos expresado el peso de los tumores de los pacientes, pero en una base de datos en centímetros y en la otra en milímetros.

2.1.1 Clasificación de los Sistemas de Integración

En este apartado se tratará el tema de la clasificación de los sistemas de integración de Bases de Datos. Como en muchas otras áreas existen distintos criterios para analizar las clasificaciones. En esta área en concreto se distinguen dos:

2.1.1.1 Criterio 1: Topología

Dependiendo de la topología se distinguen dos enfoques: centralizado y federado.

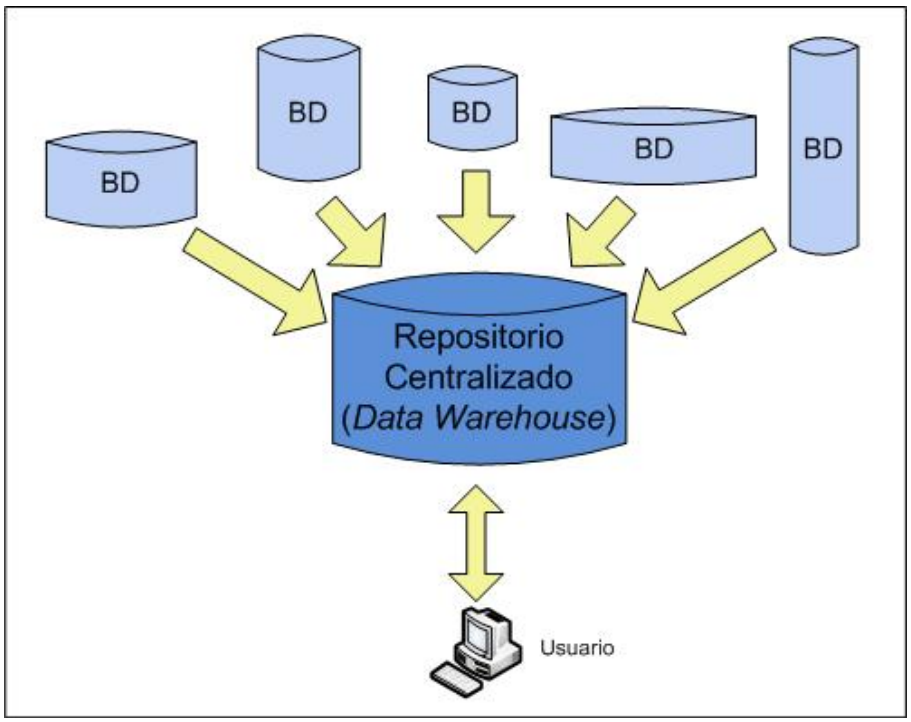

Fig 2-1 Representación gráfica de un *Data Warehouse*

El enfoque centralizado se basa en almacenar todos los datos en un repositorio central llamado *Data Warehouse* [6]. Los usuarios finales accederían a los datos de ese repositorio. El *Data Warehouse* tiene su propio modelo de datos el cual es independiente de las Bases de Datos originales, por lo que antes de almacenar los datos en el repositorio, éstos deberán ser transformados a un formato único. Esto permite que las consultas del usuario obtengan su respuesta rápidamente, ya que todos los datos se encuentran en la misma zona. Sin embargo también tiene algunas desventajas, como la posibilidad de inconsistencias en los datos (los cambios en las fuentes de datos

originales necesitan tiempo para actualizarse en el repositorio central), el elevado coste de mantener el repositorio integrado (y los datos por duplicado) y la necesidad de espacio adicional, ya que el *Data Warehouse* no deja de ser una nueva Base de Datos. La figura Fig 2-1 muestra una representación de la integración en un *Data Warehouse*.

Hay que decir que este tipo de enfoque no siempre es posible, pues existen muchas organizaciones que quieren mantener el control sobre cómo se consulta su información (por ejemplo vendedores de contenidos online) y además puede no ser factible el proporcionar copias periódicas completas de la información de las fuentes al repositorio central.

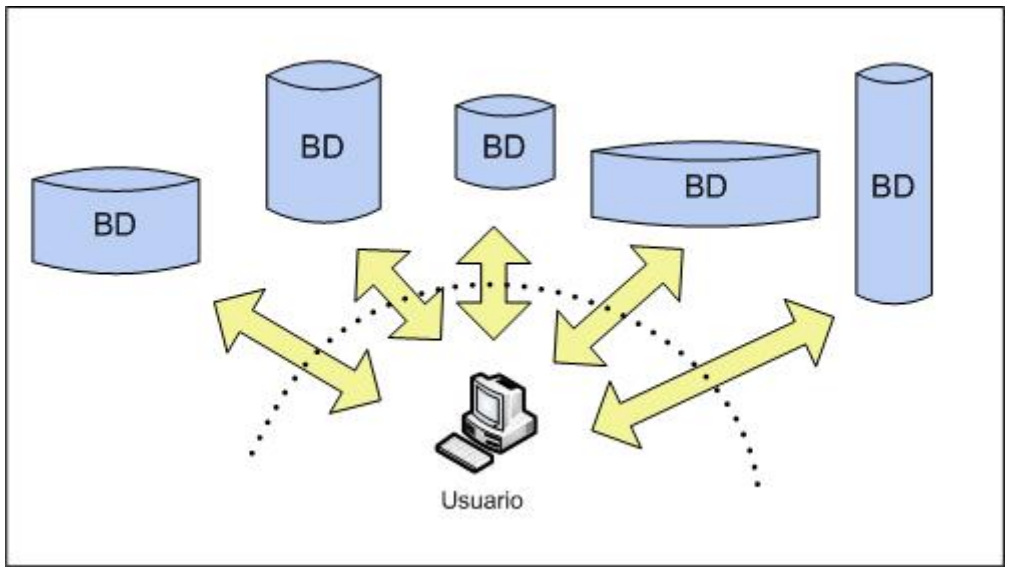

Fig 2-2 Representación gráfica del enfoque federado

Por otra parte, el enfoque federado (Fig 2-2) no se basa en un repositorio central sino que deja los datos en las fuentes originales. Es el enfoque más usado hoy en día, pues soluciona los problemas del enfoque centralizado. El enfoque federado fue presentado por [7], el cual lo llamó Sistema de Base de Datos Federado (FDBS). En este tipo de sistema, las Bases de Datos son autónomas y sus operaciones locales no dependen del FDBS; la integración se realiza en el momento previo a presentar los resultados al usuario final. Se elimina la necesidad de tener un repositorio central con los datos, manteniendo sólo un esquema global o un modelo virtual del conjunto de fuentes de datos. Por supuesto, esto implica que el tiempo de acceso será mayor, pero se resuelven los problemas del acceso centralizado.

2.1.1.2 Criterio 2: Resolución de Heterogeneidad.

Un sistema de Bases de Datos heterogéneas o *Heterogeneous Database System* (HDBS) consiste en “la creación de modelos computacionales que ofrezcan una interfaz uniforme de consultas a datos recolectados y almacenados en múltiples Bases de Datos heterogéneas”[2]. Estos sistemas son distintos de los sistemas de Bases de Datos distribuidos (DDBS), los cuales están más integrados y coordinados, pero sin embargo son más restrictivos, ya que generalmente las fuentes de datos deben usar el mismo modelo de datos, lenguaje de consultas e incluso SGBD. Los HDBS se caracterizan porque sus fuentes de datos pueden utilizar distintos modelos de datos, lenguajes de consultas, terminologías o distintos esquemas para representar la misma información [8]. De esta forma, las fuentes de datos son autónomas en cuanto a que las organizaciones responsables de cada repositorio pueden controlar el acceso y manipular los datos independientemente del HDBS en el que se incluyen. Algunas características de los HDBS son:

- Llevar a cabo la integración de Bases de Datos heterogéneas a partir de modelos de datos ya creados, y sin necesidad de grandes cambios en las fuentes de datos originales.
- El acceso a la información debe ser transparente al usuario: no debe tener necesidad de que los usuarios y/o aplicaciones que lo utilicen conozcan a existencia, localización, esquema o modo de acceso de las bases de datos locales.
- Permitir que las fuentes de datos sean gestionadas y actualizadas independientemente, sin que dichos cambios modifiquen el diseño del HDBS.
- No debe requerir que los usuarios tengan permisos de escritura sobre las fuentes originales.

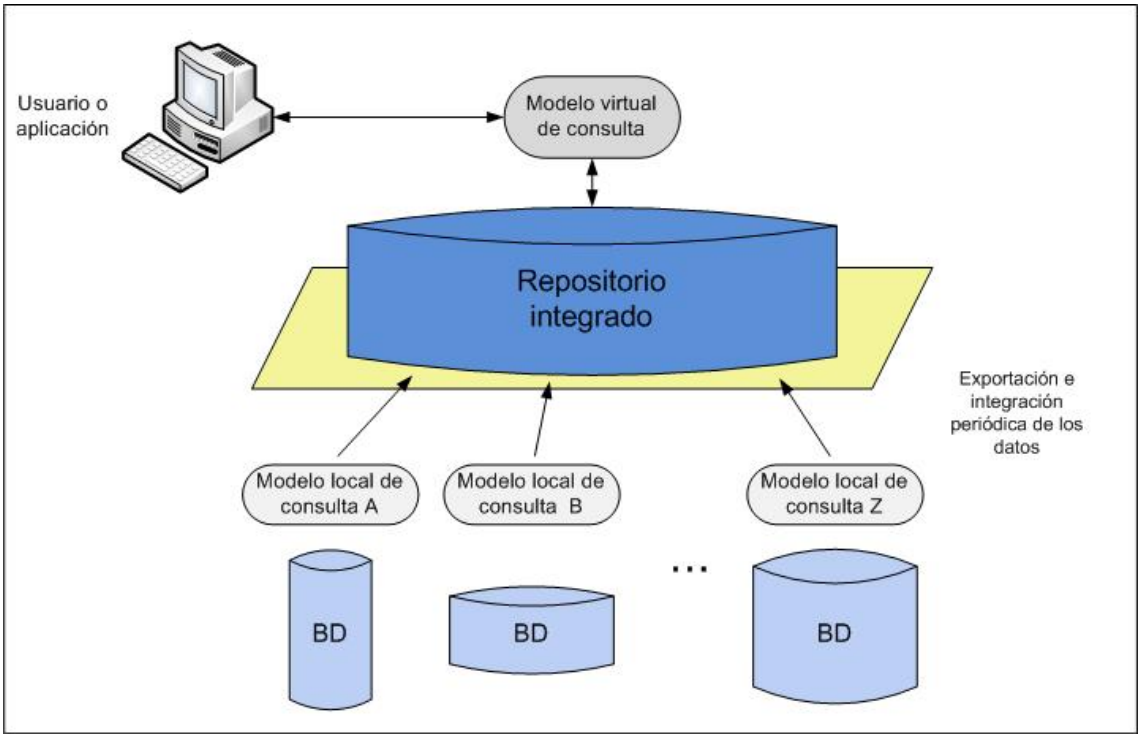

Fig 2-3 Enfoque de traducción de datos

Existen diversos enfoques para la integración de fuentes de datos heterogéneas que han sido caracterizados por la comunidad científica. Entre ellos, los más destacados son tres: enlazado de información, traducción de datos y traducción de preguntas.

En el enlazado de información, los sistemas vinculan todas las unidades de información relacionada a través de punteros o enlaces estáticos. La implementación de estos sistemas suele ser bastante sencilla, sin embargo contempla una serie de limitaciones como la unidireccionalidad de la mayoría de los *links* o la imposibilidad por parte del usuario de realizar consultas, las cuales se encuentran muy limitadas por los enlaces. Esta aproximación no tiene una gran aceptación aunque la *World Wide Web* está basada en ella y da soporte a importantes bases de datos biomédicas disponibles en la Web como MEDLINE, PDB, PROSITE y SWISS-PROT.

En los sistemas de traducción de datos (Fig 2-3), como su propio nombre indica, se traducen los datos desde el formato original en el que están almacenados hasta un formato de datos común; posteriormente a la transformación se crea un repositorio global con los datos en el formato unificado, contra el que se podrán lanzar consultas de las que se obtendrán los resultados unificados. Esta transformación puede realizarse de

forma manual o automática, y puede transformarse directamente a un formato común o a través de un paso intermedio en el que los datos se almacenen en un formato estándar. Aunque este tipo de sistemas resuelva algunos de los problemas del método anterior, produce una duplicación de los datos, lo cual incrementa las posibilidades de que los datos no estén actualizados al llegar al usuario. Además, este tipo de sistemas es difícil de mantener, verificar o modificar, incrementando la posibilidad de error, sobre todo cuando las fuentes de datos varían. Entre este tipo de sistemas se pueden citar IGD[9] y SRS[10].

Finalmente, en la alternativa de traducción de consultas, se construyen repositorios virtuales que no contienen los datos físicamente, manteniéndose éstos en las fuentes de datos originales. Cada vez que se lanza una consulta contra el sistema:

En este tipo de enfoque, cuando se recibe una consulta sobre el sistema de integración:

1) La consulta se reformula en una serie de sub-consultas sobre las fuentes de datos originales.
2) El sistema ejecuta las sub-consultas sobre las fuentes en tiempo real, recogiendo de cada fuente los datos necesarios para contestar a la consulta del usuario.
3) El sistema integra los sub-resultados obtenidos de cada fuente.
4) Devuelve al usuario el resultado integrado.

Este tipo de sistemas (Fig 2-4), genera un incremento del coste computacional al tener que traducir cada una de las consultas generadas por los usuarios y tener que esperar tanto por la devolución de los resultados como por la integración de los mismos. Además, si las fuentes de datos no tienen una interfaz para poder ser accedidas, esta estrategia no es viable. Sin embargo, con este enfoque se resuelve el problema de la actualización de los datos y el que no tengan que ser enviados periódicamente, como generalmente sucede en el campo de la biomedicina.

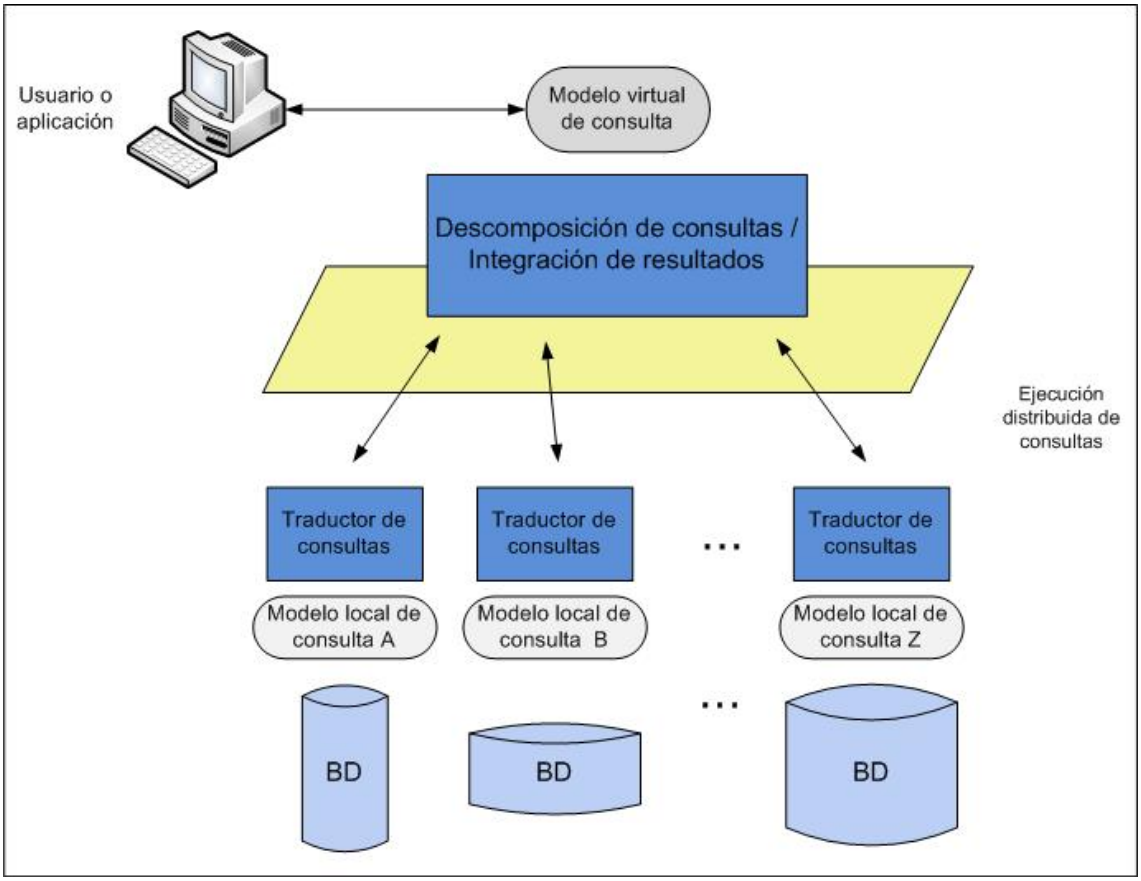

Fig 2-4 Enfoque de traducción de consultas

A su vez, esta alternativa de integración posee cuatro variantes [2]:

- Enfoque basado en mediación pura: los sistemas que utilizan este enfoque, realizan la integración a través de una serie de mediadores [11], encargados de transformar los datos procedentes de distintas fuentes a petición del usuario. Estos mediadores, almacenan la información necesaria para efectuar las transformaciones necesarias y devolvérselas al usuario, pero son menos intuitivos que los siguientes enfoques. El proyecto BACIIS implementa un sistema característico de mediación pura.

Métodos		Características	Ventajas	Inconvenientes	Ejemplos
Enlazado de información		Las entidades de información (sitios Web, instancias de una BD, etc) relacionadas están enlazadas mediante enlaces estáticos.	Sencillez de uso. El usuario sólo ha de utilizar enlaces.	La mayoría de los enlaces son unidireccionales. Consultas limitadas o predefinidas por los enlaces estáticos.	Sistemas basados en hipertexto (Internet). Bases de Datos basadas en el Web (MEDLINE, PDB, Prosite)
Traducción de datos		Los datos provenientes de distintas Bases de Datos se transforman en un formato común y se almacenan en un repositorio centralizado.	Alto rendimiento y control de datos (seguridad confiabilidad, etc)	Si se modifican los datos fuente es necesario actualizar el repositorio centralizado. Las traducciones almacenadas en los algoritmos producen problemas de mantenimiento y verificación.	Almacenes de datos o *Data Warehouses*
Traducción de consultas	Mediación pura	Los mediadores y envoltorios se usan para ejecutar las consultas de los usuarios	Contienen la información necesaria para recuperar la información y presentársela al usuario.	Es un enfoque poco intuitivo para los usuarios, pues la información de integración está codificada en los mediadores.	BACIIS
	Esquemas conceptuales simples	Existe una única conceptualización global que contiene virtualmente toda la información de las fuentes de datos. Las consultas se construyen siguiendo este esquema.	Los resultados son devueltos al usuario a nivel conceptual, como instancias de entidades que pertenecen a un esquema conceptual.	Cualquier cambio en el sistema implica un cambio en el modelo global. Sólo se puede aplicar si todos los datos provienen de un dominio común.	TSIMMIS, SIMS, ARIADNE, Garlic, DiscoveryLink, TAMBIS.
	Esquemas conceptuales múltiples	Existen distintos esquemas virtuales que describen la semántica de cada BD integrada en el sistema.	Resuelve el principal problema del enfoque conceptual simple. Las consultas pueden expresarse en términos de ontologías específicas de cada dominio.	No se puede asumir que esquemas individuales compartan el mismo vocabulario. Es complicado encontrar conceptos semánticamente similares o equivalentes en varios esquemas conceptuales. Se necesitan enlaces entre entidades similares de distintos esquemas virtuales.	OBSERVER
	Enfoque híbrido	Se usa un vocabulario común para construir ontologías que actúen como esquemas conceptuales que representan la semántica de cada fuente de datos	El sistema es más intuitivo para los usuarios debido al uso de ontologías. Los enlaces entre conceptos similares vienen dados por las relaciones de las ontologías.	Los esquemas conceptuales deben desarrollarse utilizando una misma ontología de dominio.	MECOTA, BUSTER, SEMEDA.

Fig 2-5 Comparación entre distintos enfoques de integración de datos [12]

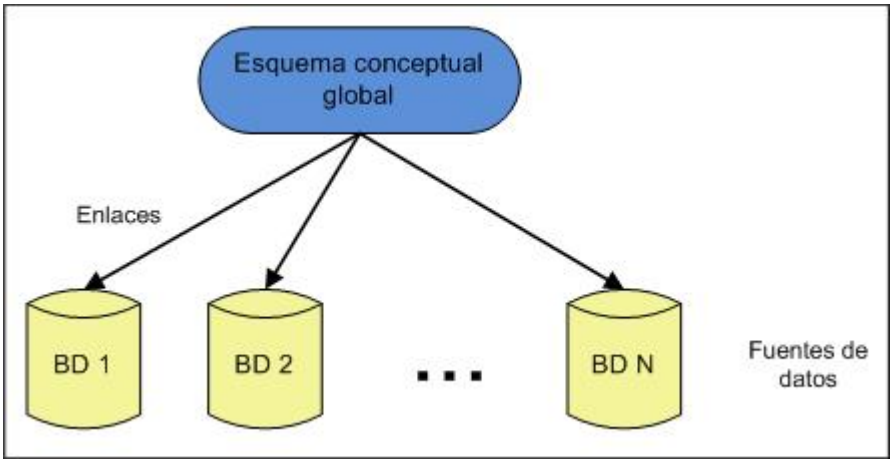

Fig 2-6 Enfoque basado en un esquema conceptual global

- Enfoque basado en un esquema conceptual global (Fig 2-6): este tipo de sistemas utilizan un esquema global, enlazado a las distintas fuentes y que suministra un vocabulario común. Dicho vocabulario se emplea en la especificación de la semántica de las fuentes a integrar y puede ser la combinación de varios esquemas conceptuales. El sistema TAMBIS está basado en este enfoque. Sin embargo, este enfoque tiene algunas desventajas: la creación de un esquema conceptual global no es siempre la mejor opción incluso para fuentes de un mismo dominio, ya que estas fuentes pueden ofrecer distintas vistas de un mismo dominio; también, si una vez establecido el esquema global y los enlaces con las fuentes, es necesario realizar alguna variación, los cambios en el esquema global podrían implicar cambios en los enlaces con las fuentes.
- Enfoque basado en múltiples esquemas conceptuales (Fig 2-7): las desventajas existentes en los esquemas globales, han llevado al desarrollo de los sistemas basados en múltiples esquemas conceptuales. En este tipo de enfoque, cada una de las fuentes de datos está descrita por su propio esquema conceptual y no tienen por qué compartir el mismo vocabulario, así, cada uno de los esquemas locales se puede desarrollar independientemente de los demás, facilitando futuras modificaciones. Se introduce un nuevo mecanismo de enlazado entre los esquemas para poder compararlos. Se trata de unos enlaces inter-esquemas que identifican las relaciones semánticas entre conceptos. En la práctica, sin embargo, estos enlaces son muy complicados de definir debido a la gran heterogeneidad

semántica que se puede dar en los sistemas de bases de datos heterogéneas. El proyecto OBSERVER es un ejemplo de este enfoque.

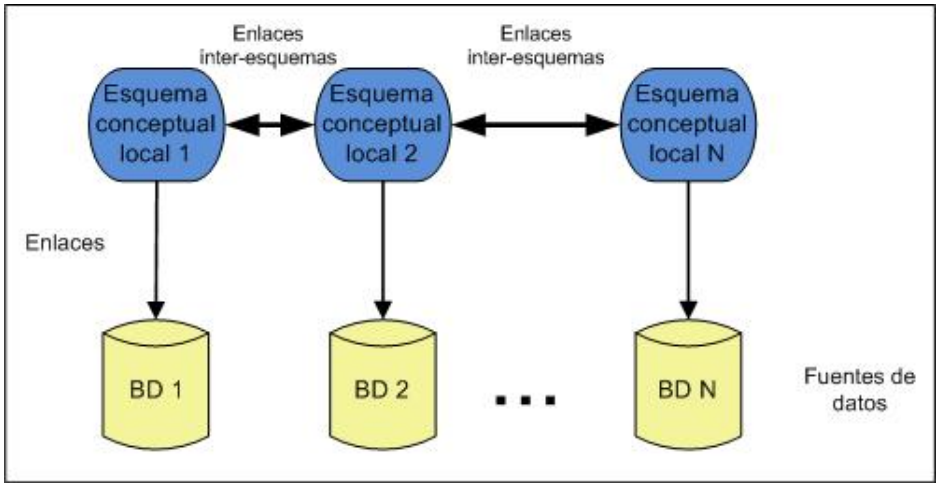

Fig 2-7 Enfoque basado en múltiples esquemas conceptuales

- Enfoque híbrido (Fig 2-8): este tipo de enfoque surgió para solucionar los inconvenientes de los dos enfoques anteriores. En este caso, cada fuente describe su propio esquema, pero se utiliza un vocabulario común para que los distintos esquemas conceptuales sean fácilmente comparables. Normalmente, los esquemas conceptuales suelen estar implementados usando ontologías, lo que aumenta la facilidad de comprensión por parte del usuario además de su reutilización. Al compartir el mismo vocabulario entre ontologías se facilita su unificación, pudiendo prescindir de los enlaces inter-esquemas. El proyecto BUSTER es un ejemplo de sistema que sigue un enfoque híbrido.

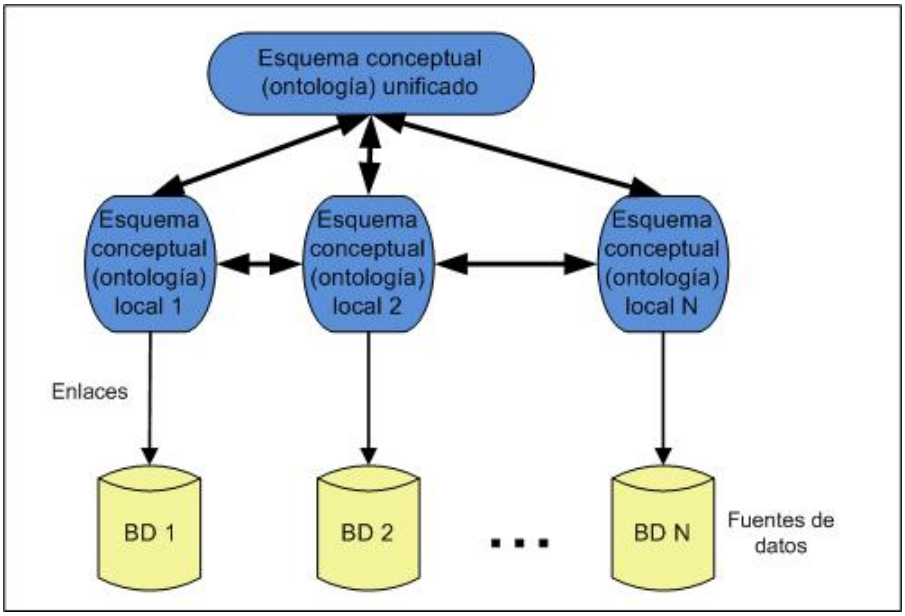

Fig 2-8 Enfoque híbrido

En la tabla de la figura Fig 2-5 se muestran las relaciones existentes entre todos estos enfoques.

2.1.2 Uso de ontologías en la integración de datos

Las ontologías pueden llegar a ser realmente útiles en la integración de datos, especialmente en cubrir los huecos sintácticos y semánticos existentes entre distintas fuentes de datos [13].

En este escenario, actualmente la integración de Bases de Datos está evolucionando hacia los enfoques híbridos y concretamente hacia aquellos basados en ontologías. Dichas ontologías facilitan el enlazado entre elementos pertenecientes a una base de datos y conceptos de un vocabulario compartido. Por ejemplo, puede darse el caso de que varias Bases de Datos contengan el mismo concepto, pero representado con distintos nombres; las ontologías se utilizarían para enlazar esos nombres al mismo concepto.

El uso de ontologías dota a las aplicaciones de un vocabulario compartido común (conceptos) para representar la información de las fuentes de datos, además de permitir definir relaciones entre conceptos (roles). Usando ontologías, tanto los conceptos como

los roles pueden ser usados para realizar consultas más complejas y recuperar de forma precisa la información en la que esté interesado el usuario.

Actualmente, la integración basada en ontologías es un área de investigación muy activa, la cual tiene varios nombres dependiendo del objetivo del sistema propuesto: mediación semántica, mediación conceptual, integración semántica de datos, etc.

2.1.3 Mediación semántica

En biomedicina, las distintas fuentes de datos existentes suelen funcionar de forma independiente y autónoma, de forma que están sujetas a una alta heterogeneidad de formatos, son actualizadas muy frecuentemente y pocas de ellas cuentan con un modelo de datos o un esquema estructurado que facilite la integración en un repositorio centralizado. Además, muchas de ellas no están implementadas con un SGBD, sino que son un conjunto de ficheros o páginas Web accesibles públicamente. Por tanto, la forma idónea para llevar a cabo la integración de dichas fuentes es mediante la implementación de un sistema de mediación o una arquitectura *wrapper*/mediador.

En un sistema de mediación, la integración se consigue usando un middleware dividido en dos partes:

- *Wrapper* (envoltorio): se sitúa sobre cada fuente de información actuando como un software que transforma los datos de una fuente de datos a un modelo de datos común a todas las fuentes. Es un traductor bidireccional: toma una petición del mediador, la transforma al lenguaje de la fuente de datos, obtiene los resultados y los traduce de vuelta al lenguaje del mediador. Resuelven la heterogeneidad sintáctica.
- Mediador: entidad centralizada encargada de recoger las consultas de los usuarios, determinar las fuentes de datos afectadas por las consultas, descomponer cada consulta en sub-consultas, dirigir cada una de ellas a la fuente de datos correspondiente, recoger los resultados de las consultas individuales, integrarlos y presentarlos al usuario

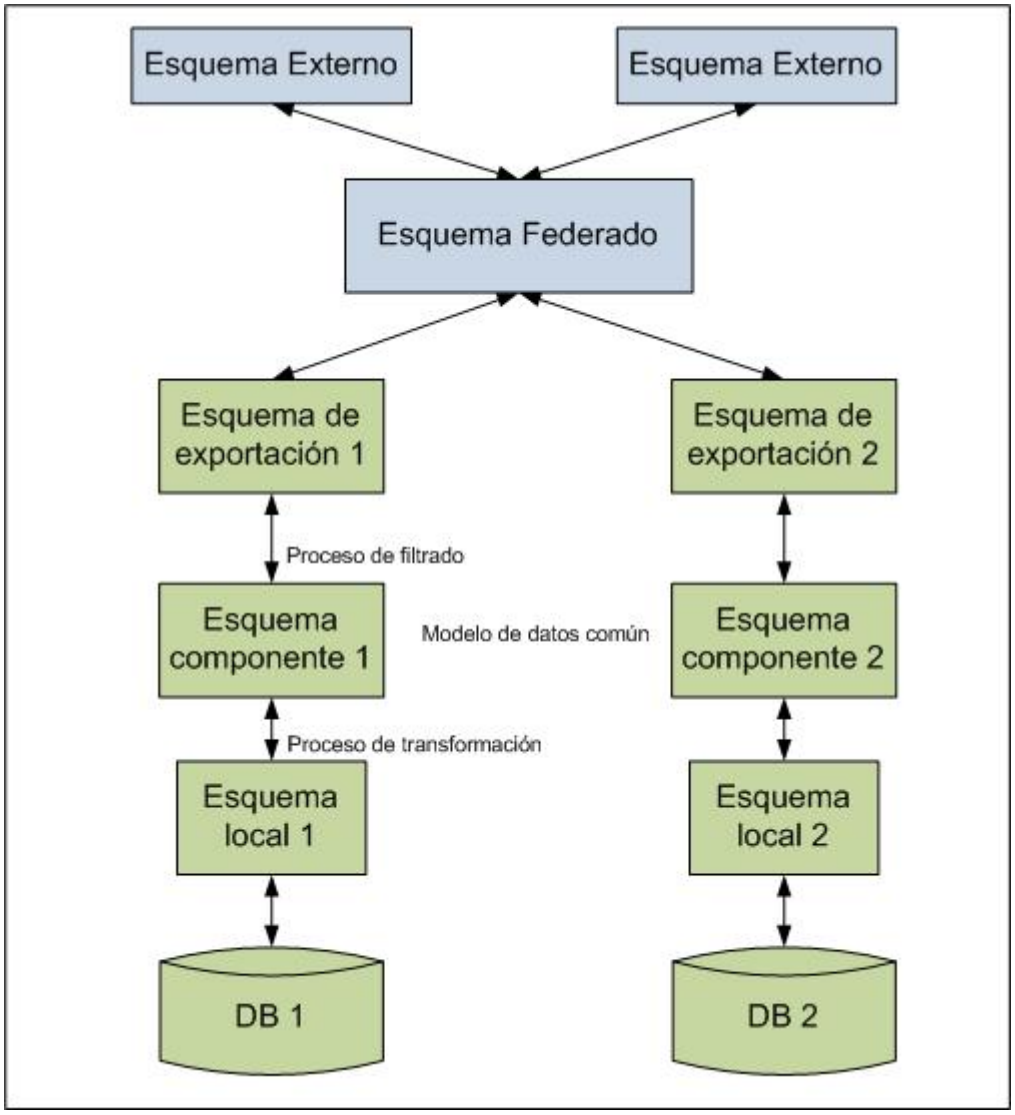

Fig 2-9 Arquitectura de cinco niveles basada en mediadores

En la mediación semántica se adopta una arquitectura de cinco niveles [14], donde existen distintos tipos de esquemas, como se puede ver en la figura Fig 2-9:

- Esquema local: es el esquema de la fuente de datos.
- Esquema componente: es el derivado de trasladar el esquema local a un modelo de datos común.
- Esquema de exportación: representa un subconjunto del esquema componente que estará disponible para el sistema de Bases de Datos federado.

- Esquema federado: consiste en una integración de múltiples esquemas de exportación. Incluye también la información de la distribución de datos que se genera cuando se integran los esquemas de exportación.
- Esquema externo: define un esquema para un usuario y/o aplicación, que puede ser usado para especificar un subconjunto de la información del esquema federado.

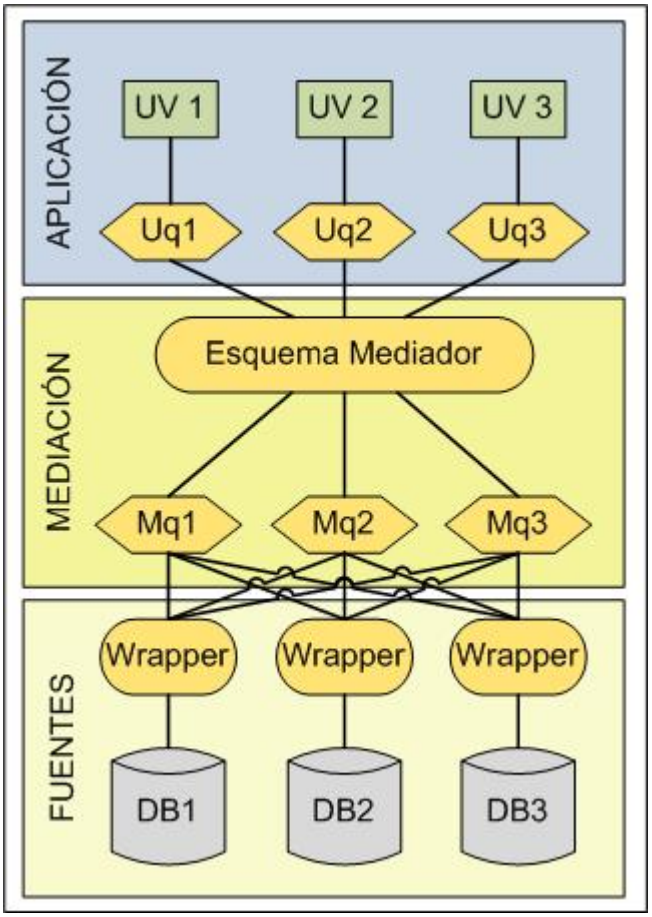

Fig 2-10 Arquitectura genérica basada en mediadores

Así, la capa de mediación se encarga tanto del esquema federado como de los esquemas de exportación, mientras que la capa *wrappers* controla los esquemas componente y local de cada fuente de datos, y el acceso a dicha fuente.

Como se ha visto, en la arquitectura basada en mediadores existen distintas capas (Fig 2-10) [15]:

- La capa “fuente”: formada por los *wrappers* (que resuelven la heterogeneidad sintáctica y del sistema de las fuentes de datos) y las fuentes de datos.
- La capa “mediación”: que realiza la integración incluyendo un esquema federado y una serie de *mappings* (Mq en la figura), donde se establecen las relaciones entre el esquema del mediador y el esquema de los *wrappers*.
- La capa “aplicación”: formada por las vistas de los usuarios (UV en la figura), que son los datos usados en las aplicaciones, y las consultas generadas por los usuarios (Uq).

Respecto a la descripción de las fuentes conforme al esquema global del mediador, existen dos enfoques (Fig 2-11):

- Global as View (GaV): se trata de un enfoque Top-Down donde el esquema del mediador es una vista de las fuentes de datos.
- Local as View (LaV): se trata de un enfoque Bottom-Up donde las fuentes son vistas del esquema del mediador.

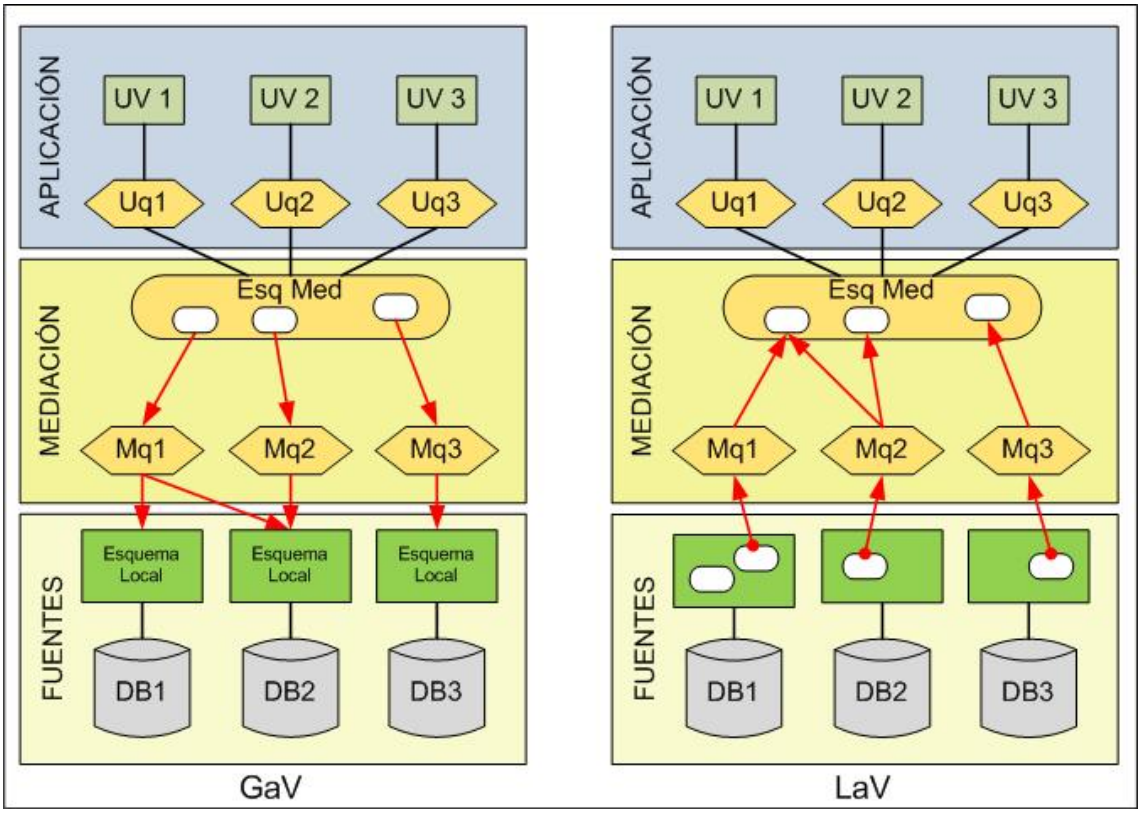

Fig 2-11 Enfoque GaV y enfoque LaV

2.1.4 Herramientas

En esta sección se comentarán algunas de las herramientas nombradas en este capítulo.

2.1.4.1 El proyecto BACIIS

En el proyecto BACIIS [16] (*Biological and Chemical Integrated Information Systems*) desarrollaron un sistema de integración de Bases de Datos Web sobre química y biología. Este sistema permite realizar búsquedas en las Bases de Datos subyacentes, como OMIM [17], PDB [18] o Prosite [19] y muestra mediante su interfaz Web muestra páginas con los resultados.

La diferencia fundamental entre BACIIS y otros sistemas consiste en que BACIIS almacena información sobre los tipos de entrada y salida de cada fuente que esté integrada en el sistema. El funcionamiento de este sistema está basado en la información relativa a las fuentes obtenida por un mediador. Este mediador genera un plan de ejecución de la consulta seleccionando las Bases de Datos más apropiadas. A continuación, los *wrappers* devuelven los resultados de las Bases de Datos para que el mediador los integre, de acuerdo con las relaciones establecidas en una ontología de dominio (Fig 2-12).

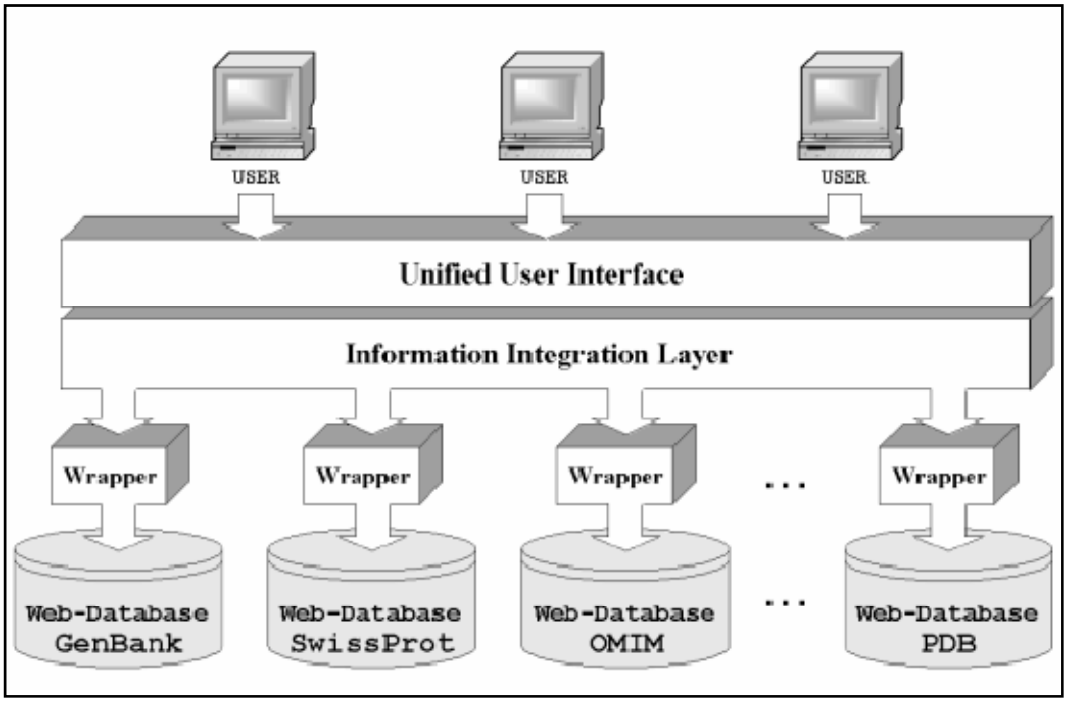

Fig 2-12 Arquitectura de BACIIS [16]

BACIIS se fundamenta en el enfoque de traducción de consultas mediante mediación pura. Su núcleo principal posee una ontología que, además de guiar a los usuarios a la hora de crear consultas, sirve para consensuar la terminología biológica y química. Sin embargo, a pesar de utilizar esta ontología de dominio, su arquitectura basada en mediadores no realiza una integración a nivel conceptual.

2.1.4.2 El proyecto TAMBIS

El sistema TAMBIS (*Transparent Access to Multiple Bioinformatics Information Sources*) [20] se encarga de proporcionar a los usuarios un acceso sencillo y transparente a diferentes servicios de información biológica, construyendo una capa homogeneizadora sobre todas las fuentes de datos. Este sistema está fundamentado en el enfoque de mediación guiada por modelos de dominio globales; establece relaciones de correspondencia entre los objetos del esquema físico de las fuentes y una ontología global.

El sistema TAMBIS posee una arquitectura en la que se encuentran presentes los siguientes componentes:

- Una ontología de términos biológicos.
- Una interfaz de desarrollo de consultas guiada por conocimiento.
- Un modelo de servicios para crear correspondencias entre la ontología y los esquemas de las fuentes de datos.
- Un proceso de escritura de transformación de consultas.
- Un servicio *wrapper* para tratar con las fuentes externas.

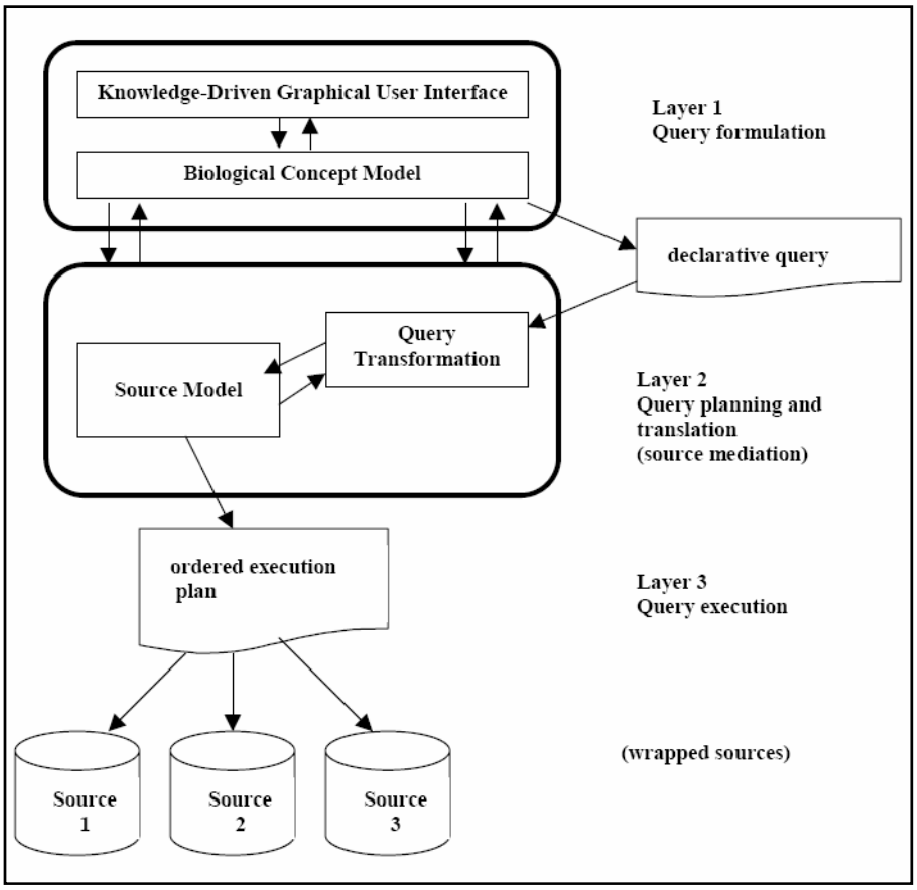

Fig 2-13 Arquitectura de TAMBIS [20]

2.1.4.3 El proyecto OBSERVER

El sistema OBSERVER (*Ontology Based System Enhanced with Relationships for Vocabulary hEterogenity Resolution*) [21], trata la consulta de datos en fuentes fuentes distribuidas y heterogéneas. Está basado en el enfoque de traducción de consultas con múltiples esquemas virtuales. Cada fuente integrada en OBSERVER tiene una ontología que señala las correspondencias entre objetos físicos y conceptuales.

Este sistema posee una interfaz de consulta basada en ontologías, que serán usadas por el usuario para construir las consultas. El sistema analiza cada una de las consultas y la envía a las fuentes de datos asociadas a la ontología elegida. Los *wrappers* se encargan de la tarea de traducir las consultas al formato de cada fuente y más tarde envían los resultados al mediador, que los recopila e integra para devolvérselos al usuario. La arquitectura del sistema OBSERVER se muestra en la figura Fig 2-14:

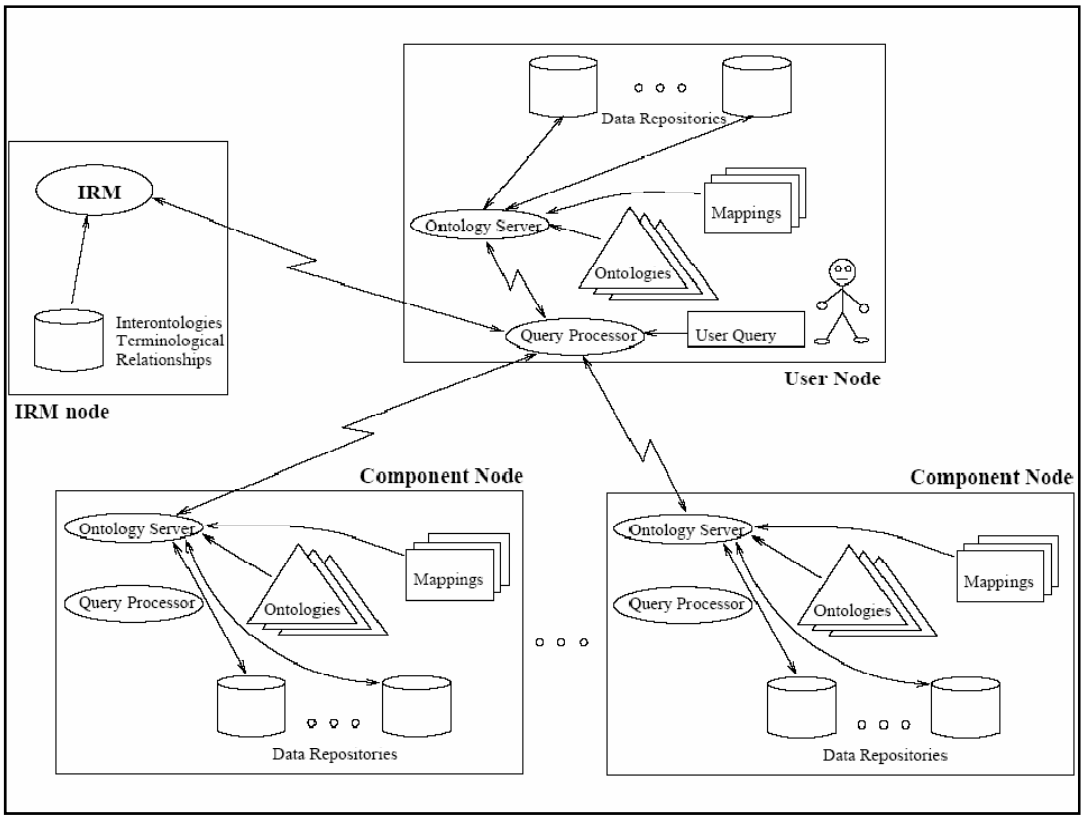

Fig 2-14 Arquitectura de OBSERVER [21]

2.1.4.4 El proyecto BUSTER

El sistema desarrollado dentro de este proyecto, BUSTER (*Bremen University Semantic Translator for Enhanced Retrieval*) [22], consiste en un mediador para la integración semántica de datos. El enfoque que sigue es el de traducción de consultas, bajo la perspectiva híbrida.

BUSTER está formado por dos componentes principales:

- BUSTER/Q: herramienta para la recuperación inteligente de información.

- BUSTER/SI: herramienta para la integración semántica de las fuentes de datos heterogéneas.

BUSTER no reutiliza ontologías como el sistema OBSERVER, sino que lo que hace es crear una nueva para cada fuente, utilizando los conceptos, atributos y relaciones existentes en una ontología de dominio global. De esta forma los objetos de cada ontología crean correspondencias con el mismo objeto conceptual.

El usuario construye la consulta con objetos de la ontología por medio de una interfaz. A partir de ahí el proceso de resolución de la consulta consiste en:

- Encontrar las fuentes que contengan información relevante deseada.
- Enviar las consultas a cada fuente pasando por los *wrappers*, que transforman la consulta al lenguaje específico de la fuente.
- Los *wrappers* envían los datos resultantes de cada fuente al mediador.
- El mediador unifica los resultados para presentarlos al usuario como instancias de objetos pertenecientes a la ontología de dominio.

2.1.4.5 OntoFusion

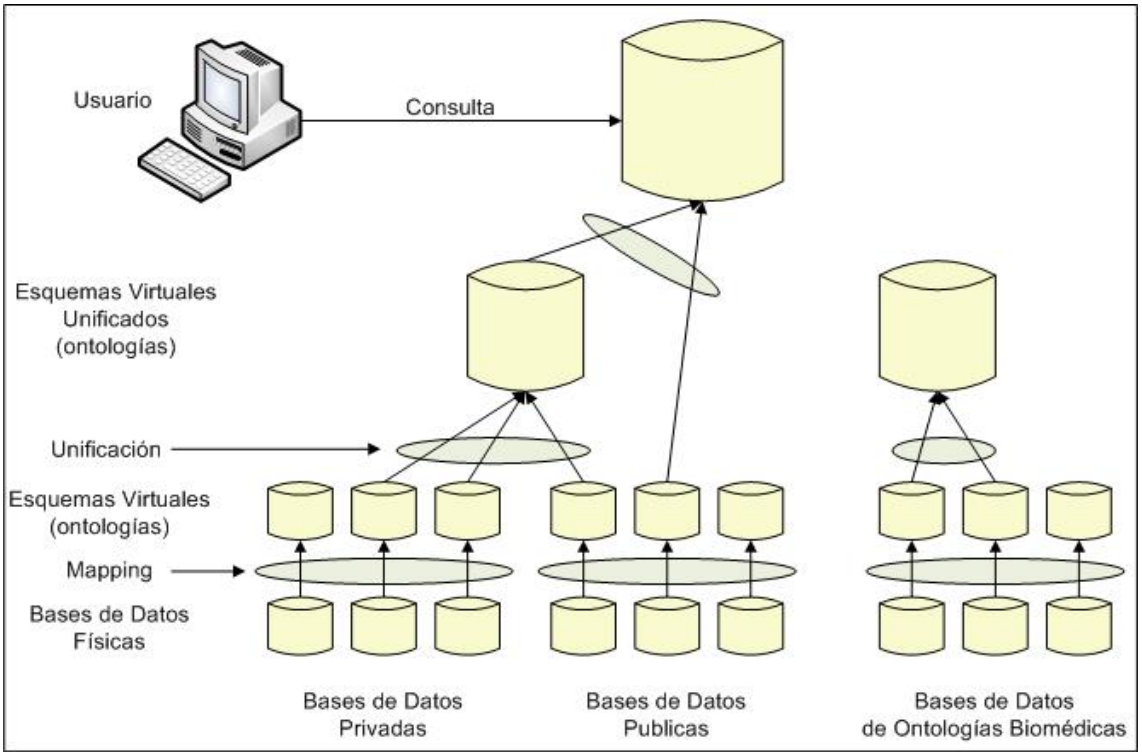

Fig 2-15 OntoFusion

El sistema OntoFusion [23] surgió de un proyecto financiado por la Comisión Europea llamado INFOGENMED. Este proyecto pretendía proporcionar una solución para llevar a cabo la integración de fuentes de datos estructuradas (principalmente Bases de Datos relacionales) orientadas fundamentalmente a información clínica y genética.

OntoFusion se basó en el enfoque basado en mediación, con una aproximación híbrida. El sistema (Fig 2-15) posee una ontología por cada fuente de datos, la cual describe la semántica de la fuente. Estas ontologías, junto con otra información adicional se denominan repositorios virtuales. Como se muestra en la anterior figura, existen dos procesos fundamentales: *mapping* (establecimiento de correspondencias entre el esquema físico de la base de datos y un esquema conceptual) y unificación (unión de varios esquemas virtuales de varias fuentes de datos en un esquema virtual unificado).

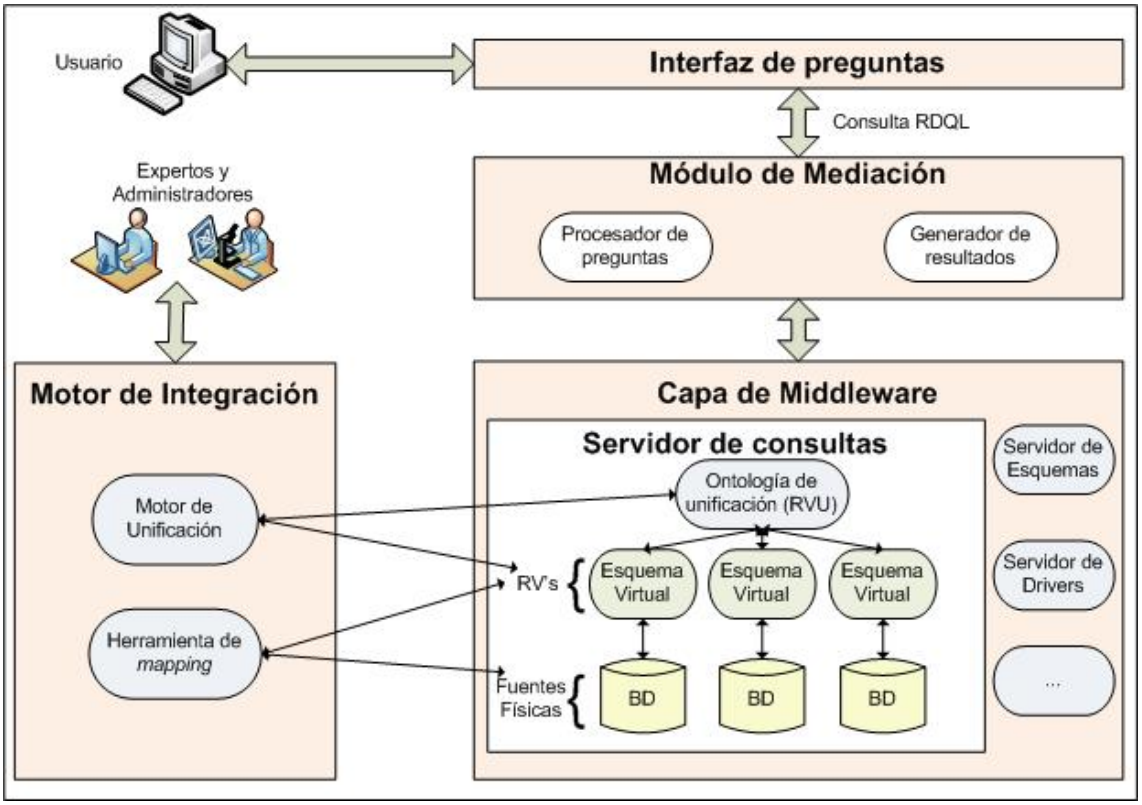

Fig 2-16 Arquitectura de OntoFusion

La arquitectura de OntoFusión (Fig 2-16) consta de cuatro componentes principales:

- El interfaz de realización de preguntas: es un interfaz gráfico Web, basado en applets de Java y que posibilita la visualización y navegación a través de la jerarquía de repositorios virtuales del sistema. En esta interfaz, se selecciona un repositorio, y ésta simula los elementos que tiene, dando la impresión de que se trabaja directamente con los repositorios locales.
- El módulo de mediación: los mediadores en OntoFusion están presentes a nivel de los repositorios virtuales de unificación y de los repositorios virtuales de *mapping*. Podría decirse que éstos equivalen a los *wrappers* del modelo de integración que sigue un enfoque de traducción de consultas. El proceso de resolución de consultas en OntoFusion es:
    - El usuario lanza una consulta sobre un repositorio virtual concreto.
    - Si el repositorio es un repositorio virtual unificado, actúa como mediador determinando si los repositorios subyacentes poseen información

relevante. A aquellos que si la tengan les envía las consultas en el formato adecuado.

  - Las consultas “descienden” por los repositorios virtuales unificados hasta que dan con repositorios virtuales de *mapping*.
  - Estos repositorios virtuales de *mapping* actúan como *wrappers*, pues traducen las consultas a los formatos nativos de las fuentes de datos.
  - Cuando los resultados ya están presentes en los *wrappers*, son propagados hacia el usuario, de forma que, a medida que van pasando por los repositorios virtuales unificados, los resultados se van integrando.
  - Se devuelven los resultados unificados al usuario, de forma gráfica.

- La capa de middleware: formada por un conjunto de servicios (servicios Web Java) que hacen posible que OntoFusion funcione de forma distribuida. Estos servicios, además de abstraer al sistema de la complejidad de las redes subyacentes, sistemas operativos y lenguajes de programación, se encargan de proporcionar a los administradores del sistema un método para conectar nuevas fuentes de datos al sistema de consultas. Esta capa también da soporte al uso del conjunto de funciones y procedimientos o API (interfaces de programación de aplicaciones). Este API ofrece los servicios de consultas para que los usuarios puedan realizarlas y obtener los resultados de las mismas.
- Motor de integración: compuesto por la herramienta de *mapping* y el motor de unificación, los cuales dan la posibilidad a los administradores de integrar fuentes estructuradas en el sistema. Esta integración se realiza de forma semiautomática siguiendo la filosofía de los repositorios virtuales, que siguen la idea de mantener sólo los metadatos de las fuentes, no los datos; los repositorios virtuales están descritos por los esquemas virtuales unificados, que son unos ficheros generados por el motor de integración. Los esquemas virtuales se crean en la operación de *mapping*, son el establecimiento de las correspondencias entre objetos físicos y objetos conceptuales. Resumiendo:
  - La herramienta de *mapping*: partiendo del esquema físico y de un modelo de dominio, permite que los administradores establezcan el esquema virtual de la fuente.

- El motor de unificación: dado un conjunto de esquemas virtuales de fuentes a integrar, construye automáticamente el esquema virtual unificado.

2.2 TECNOLOGÍAS, LENGUAJES Y ESTÁNDARES

2.2.1 Java

Java es un lenguaje de programación desarrollado por Sun Microsystems a principios de los años 90 [24]. Las aplicaciones Java están típicamente compiladas en un *bytecode*, aunque también es posible la compilación en código máquina nativo. En tiempo de ejecución, el *bytecode* es normalmente interpretado o compilado a código nativo para la ejecución, siendo posible también la ejecución directa por hardware del *bytecode* por un procesador Java.

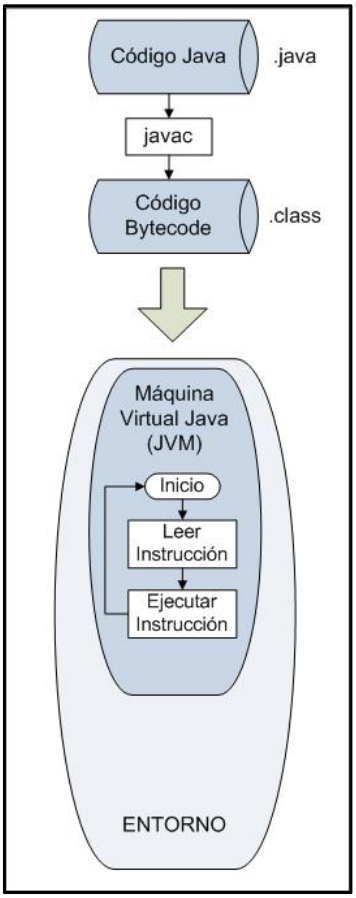

Fig 2-17 Compilación-Interpretación Java

La compañía Sun describe el lenguaje Java como “simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico”. Estas características se detallan a continuación [25]:

- Simple: Java tiene una curva de aprendizaje muy rápida. Es bastante sencillo programar applets interesantes desde el principio. Aquellas personas familiarizadas con otros lenguajes como C++ encontrarán que programar en Java resulta más sencillo, ya que es un lenguaje semejante pero del que se han eliminado ciertas características, como los punteros (el entorno de ejecución de Java tiene un recolector de basura que libera periódicamente la memoria ocupada por los objetos que no se van usar más).
- Orientado a objetos: Java se diseñó como un lenguaje orientado a objetos desde el principio. Los objetos agrupan en “estructuras encapsuladas” los datos y los métodos que manipulan dichos datos. La tendencia futura (a la que se suma este lenguaje), apunta hacia la programación orientada a objetos, especialmente en entornos complejos y/o basados en red.
- Distribuido: este lenguaje facilita la creación de aplicaciones distribuidas proporcionando una colección de clases para su uso en aplicaciones de red. Estas clases permiten abrir *sockets* y establecer y aceptar conexiones con servidores o con clientes remotos.
- Interpretado y compilado a la vez: Java es compilado pues su código fuente se transforma en una especie de código máquina, los *bytecodes* antes mencionados. Por otra parte, también es interpretado, pues los *bytecodes* se pueden ejecutar directamente sobre cualquier máquina a la cual se hayan portado el intérprete y el sistema de ejecución en tiempo real (*run-time*). Esta compilación y posterior interpretación se puede ver en la Fig 2-17.
- Robusto: Java se diseñó para crear software altamente fiable. Proporciona numerosas comprobaciones en tiempo de compilación y de ejecución.
- Seguro: dada la naturaleza de este lenguaje, en el que los applets se pueden bajar desde cualquier punto de la Red, se consideró una necesidad vital la seguridad. De esta forma, se implementaron barreras de seguridad en el lenguaje y en el sistema de ejecución en tiempo real.

- Independiente de la arquitectura: Java está diseñado para soportar aplicaciones que posteriormente se ejecutarán en los más variados entornos de red, desde Unix a Windows, pasando por Mac y estaciones de trabajo, sobre diversas arquitecturas y con sistemas operativos distintos. Éste es el punto fuerte de los *bytecodes*: es un formato intermedio, indiferente a la arquitectura, y diseñado para transportar eficientemente el código a múltiples plataformas hardware y software. El resto de problemas son resueltos por el intérprete de Java.
- Portable: la indiferencia a la arquitectura representa sólo una parte de su portabilidad. Java también especifica los tamaños de sus tipos de datos básicos así como el comportamiento de sus operadores aritméticos, de forma que los programas funcionen del mismo modo en todas las plataformas. Estas dos últimas características se conocen como la Máquina Virtual Java (JVM).
- Multitarea: hoy en día es una gran limitación las aplicaciones que sólo pueden ejecutar una acción a la vez. Java soporta la sincronización de múltiples hilos de ejecución (*multithreading*) a nivel de lenguaje, los cuales son especialmente útiles en la creación de aplicaciones de red distribuidas. De esta forma, un hilo puede encargarse de la comunicación, otro puede interactuar con el usuario mientras otro presenta una animación en pantalla y otro realiza cálculos.
- Dinámico: el lenguaje Java y su sistema de ejecución en tiempo real son dinámico en la fase de enlazado. Las clases se enlazan sólo cuando son necesitadas.

Un detalle interesante en Java es que al programar, cualquier aplicación que se desarrolle, se apoya en un gran número de clases preexistentes y, por tanto, no se parte de cero.

Este lenguaje multiplataforma permite, entre otras cosas:

- Crear programas que funcionen tanto en Navegadores como Servicios Web.
- Desarrollar software en una plataforma y ejecutarlo prácticamente en cualquier otra.
- Combinar aplicaciones o servicios que usen Java para crear servicios o aplicaciones totalmente personalizados.

2.2.2 XML

Siglas en inglés de *eXtensible Markup Language* (lenguaje de marcas extensible) [26].

XML proviene del lenguaje GML (*Generalized Markup Language*), inventado por IBM en los años setenta, que surgió de la necesidad que tenía la empresa de almacenar grandes cantidades de información. En 1986 la ISO trabajó para normalizar este lenguaje, dando lugar a SGML (*Standard Generalized Markup Language*), capaz de adaptarse a un gran abanico de problemas.

En 1989 Tim Berners Lee creó la Web, y el lenguaje HTML (*HyperText Markup Language*). Este lenguaje se definió en el marco de SGML. HTML es un lenguaje al que los navegadores Web han puesto pocas exigencias, de forma que muchas páginas Web terminan resultando caóticas, pues tienen que lidiar de una forma específica con los errores y las ambigüedades. Esto hace a las páginas Web más frágiles y a los navegadores más complejos.

Otra limitación de SGML es que cada documento pertenece a un vocabulario fijo, establecido por el DTD (*Document Type Definition*).

Se buscó definir un nuevo subconjunto SGML donde:

- Se pudieran mezclar elementos de diferentes lenguajes. Es decir, que los lenguajes fueran extensibles.
- No hubiera una lógica especial para cada lenguaje, para permitir la creación de analizadores simples.
- No se aceptara nunca un documento con errores de sintaxis.

Así, surgió XML, orientado a hacer las cosas más sencillas para los programas automáticos que necesitasen interpretar un documento.

Las características y ventajas de XML son:

- Su uso futuro en la Web mejorará la eficiencia en las búsquedas, al proporcionar cada documento XML metadatos sobre sí mismo.
- Permite proporcionar diferentes vistas sobre los datos (HTML, PDF, etc) dependiendo de quién sea el cliente.

- Facilita la integración desde fuentes de datos heterogéneas como distintas páginas Web, bases de datos, etc.
- Los documentos tienen una estructura que los hace fácilmente legibles tanto para los ordenadores como para los humanos.
- Las aplicaciones de XML son pueden ser extensibles mediante definiciones de nuevos tipos de documento (DTD).
- El analizador es un componente estándar y no es necesario crear uno específico para cada lenguaje.
- Es un lenguaje muy similar a HTML pero su función principal es describir los datos, no mostrarlos.
- XML usa etiquetas, pero sólo para delimitar las piezas de datos; la interpretación de los mismos es tarea de la aplicación que los lee.

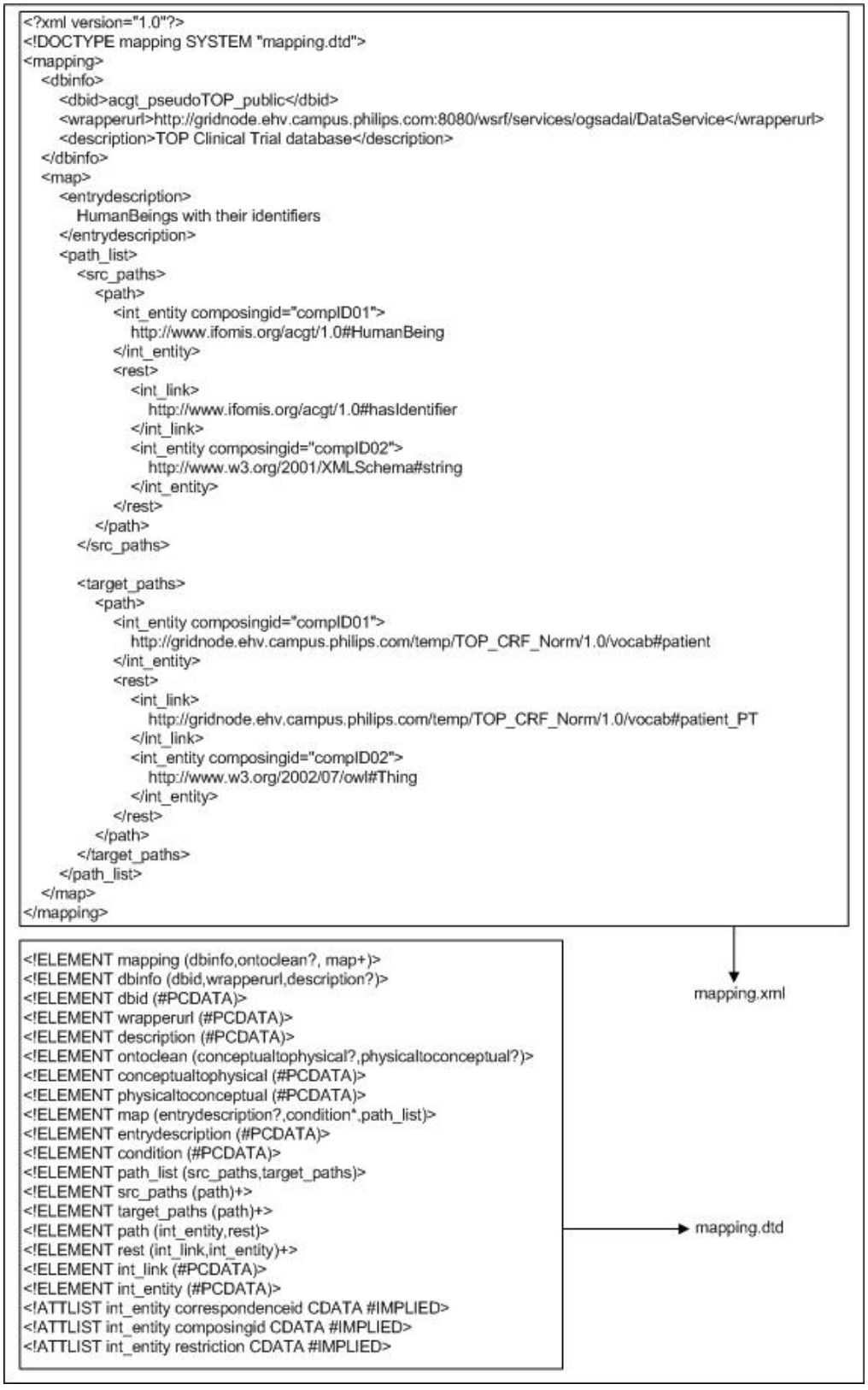

Fig 2-18 Relación entre documento XML y documento DTD

A lo largo del tiempo han surgido distintas formas de definir un documento XML [27]:

- DTD: como ya se ha mencionado, define la semántica del tipo de documento que se va a crear. Define por tanto, los elementos disponibles, las relaciones entre ellos, los atributos, posibles valores, etc. Representa la estructura válida para una clase concreta de documentos XML (Fig 2-18).
- XML SCHEMA: aunque tiene el mismo objetivo que los DTD, los XML SCHEMAS son en sí mismos documentos XML, por lo que se pueden utilizar las mismas herramientas para crear las semánticas y los documentos.
- RDF (*Resource Description Framework*): es un lenguaje de descripción del W3C (*World Wide Web Consortium*). Su objetivo principal era manejar grandes cantidades de documentos HTML o XML. El lenguaje RDF crea metadatos para un conjunto de documentos para que éstos puedan ser encontrados más tarde.
- XML Namespaces: son un método simple para distinguir entre nombres de elementos y atributos que se utilizan en documentos XML, mediante una asociación de éstos con referencias a URIs (*Uniform Resource Identifier*). Su principal objetivo de la especificación de espacios de nombre (*namespaces*) es permitir que el autor del documento le diga al analizador sintáctico qué DTD usar para el análisis de un elemento dado.

Los analizadores sintácticos (*parser* en inglés), leen el documento XML y, como mínimo, verifican que cumplen las especificaciones del lenguaje. El *parser* es la herramienta principal de toda aplicación XML, pues permite reconocer las distintas partes del documento y trabajar con ellas.

2.2.2.1 XML y Java: DOM y SAX

DOM y SAX son dos de estos analizadores sintácticos que nos permiten manejar documentos XML en Java.

La API DOM (*Document Object Model*) fue definida por el W3C. Para cada documento analizado proporciona un objeto en memoria con estructura de árbol. De forma que podremos navegar por él utilizando diversos mecanismos.

La API SAX (*Simple API for XML*) fue definida por el grupo XML-DEV. Permite analizar los documentos mediante la programación de eventos. Cada evento se va lanzando medida que se reconocen las distintas etiquetas.

2.2.3 RDF

El Marco de Descripción de Recursos (*Resource Description Framework*), desarrollado por la W3C, es una estructura de soporte definida para metadatos en la World Wide Web.

Es muy difícil automatizar cualquier cosa en la Web [28]; debido al gran volumen de información que contiene, no se puede gestionar manualmente. W3C propuso usar metadatos para describir los datos de la Web y para gestionar dicha información. Los metadatos son “datos sobre los datos”.

El objetivo principal de RDF es definir un mecanismo adecuado para describir recursos, que no cree ninguna asunción sobre un dominio de aplicación particular, ni defina, a priori, la semántica del mismo.

RDF es una base para procesar metadatos que destaca por la facilidad para habilitar el procesamiento automatizado de los recursos Web.

La base de RDF es un modelo para representar propiedades designadas y valores de propiedades. Las propiedades RDF pueden recordar a atributos de recursos y representan también la relación entre recursos. El modelo de datos básico consiste en tres tipos de objetos:

- Recursos: todo aquello descrito por expresiones RDF se denomina recurso. Los recursos se designan siempre mediante URIs.
- Propiedades: son un aspecto específico, característica, atributo o relación utilizado para describir un recurso. Las propiedades tienen su significado concreto, definen sus valores permitidos, los tipos de recursos que pueden describir y sus relaciones con otras propiedades.
- Sentencias (declaraciones, enunciados): se llama sentencia a la unión de un recurso junto con la propiedad denominada, más el valor de dicha propiedad para ese recurso. Estas tres partes se denominan respectivamente sujeto, predicado y objeto.

Dentro de RDF se puede hablar de RDF Schema (RDFS). Un archivo RDFS es un archivo RDF y por tanto sigue la misma sintaxis y la misma estructura que usa RDF (basada en XML). Es una extensión semántica de RDF. Un lenguaje primitivo que suministra los elementos básicos para la descripción de vocabularios. La primera versión se publicó en abril de 1998 por la W3C.

2.2.3.1 RDF y Java: Jena

Jena es un *framework* o marco desarrollado por HP Labs para manipular metadatos desde una aplicación Java. Este API incluye varios componentes:

- ARP: un *parser* de RDF.
- API RDF.
- API de Ontologías con soporte para OWL (*Ontology Web Language*), DAML (*DARPA AgentMarkup Language*) y RDF Schema.
- Subsistema de razonamiento.
- Soporte para persistencia.
- RDQL y SPARQL: lenguajes de consultas de RDF.

2.2.4 SPARQL

SPARQL es un acrónimo recursivo del inglés *SPARQL Protocol and RDF Query Language*. Fue desarrollado por el W3C y se trata de una recomendación para crear un lenguaje de consulta dentro de la Web Semántica.

[29] SPARQL define un lenguaje de recuperación para RDF/RDFS. Gracias a esta tecnología de consulta, las personas pueden centrarse en la información que quieren, sin tener en cuenta la tecnología de la base de datos o el formato utilizado para almacenar los mismos.

Los desarrolladores y usuarios finales son capaces de representar y utilizar los resultados obtenidos en las búsquedas en SPARQL a través de una gran variedad de información como son datos personales, redes sociales y metadatos sobre recursos digitales como música e imágenes.

[30] Este lenguaje puede usarse para expresar consultas a diversas fuentes de datos, siempre que estos datos estén almacenados en formato RDF o que se vean como RDF

por medio de algún middleware. Los resultados de las consultas SPARQL serían conjuntos de sentencias RDF.

La especificación de SPARQL consta de tres documentos:

- *SPARQL Query Language for RDF*: define la sintaxis y la semántica de las consultas, para la composición, concordancia y experimentación.
- *SPARQL Protocol for RDF*: define un protocolo para el acceso remoto entre una aplicación que emita consultas SPARQL y un servidor que le envíe los resultados.
- *SPARQL Query Results XML Format*: especifica un formato XML para los resultados de las consultas SPARQL.

Para las consultas, SPARQL usa una sintaxis que puede recordar a la de SQL [31]:

- PREFIX: define los prefijos para los espacios de nombres que se utilizarán en la consulta.
- SELECT: identifica las variables que se devolverán como resultados.
- FROM: nombra los grafos que serán consultados.
- WHERE: patrones para la consulta en forma de una lista de tripletas (dominio, relación, rango)
- FILTER: establece restricciones sobre las tripletas de la consulta.

Un ejemplo de consulta SPARQL se puede ver en la Fig 2-19

```
PREFIX acgt:     &lt;http://www.ifomis.org/acgt/1.0#&gt;
PREFIX xsd:      &lt;http://www.w3.org/2001/XMLSchema#&gt;
PREFIX bfo_span: &lt;http://www.ifomis.org/bfo/1.1/span#&gt;
PREFIX bfo_snap: &lt;http://www.ifomis.org/bfo/1.1/snap#&gt;
PREFIX ro:       &lt;http://www.ifomis.org/obo/ro/1.0#&gt;

SELECT ?diameterValue
WHERE {

 ?patient            ro:hasPart           ?nephroblastoma .
 ?nephroblastoma     acgt:hasDiameter     ?diameter .
 ?diameter           acgt:hasFloatValue   ?diameterValue .
 ?patient            a                    acgt:HumanBeing .
 ?nephroblastoma     a                    acgt:Nephroblastoma .
 ?diameter           a                    acgt:Diameter .
 ?diameterValue      a                    xsd:float .

 ?patient2                    ro:hasPart  ?highRiskNephroblastoma .
 ?patient2                    a           acgt:HumanBeing .
 ?highRiskNephroblastoma      a           acgt:HighRiskNephroblastoma .

 FILTER ( ?patient = ?patient2 )
}
```

Fig 2-19 Ejemplo de consulta SPARQL

2.2.4.1 SPARQL, Java y Bases de Datos Relacionales: d2rq

D2rq es una aplicación *wrapper* o "envoltorio" que publica en RDF los contenidos de una Base de Datos Relacional. Ha sido desarrollada por la Freie Universität Berlin.

Realiza una construcción automática del mapeo analizando la estructura de la Base de Datos [32].

La siguiente figura (Fig 2-20) muestra la estructura de un ejemplo de mapeo d2rq:

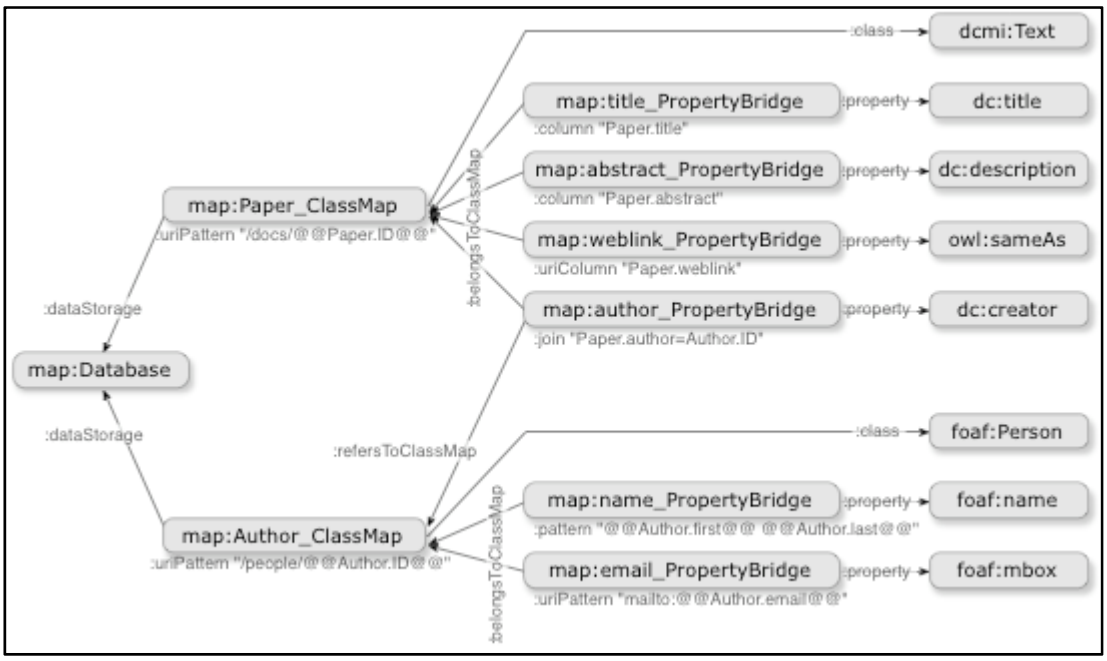

Fig 2-20 Ejemplo de mapeo d2rq

A través de d2rq se pueden realizar consultas SPARQL a un modelo d2rq. Dicho modelo d2rq podría haberse creado a partir de una Base de Datos Relacional, de forma que finalmente se podrían realizar consultas SPARQL a una Base de Datos SQL por ejemplo.

2.2.5 OWL

OWL, acrónimo del inglés *Ontology Web Language*, es un lenguaje de marcado para publicar y compartir datos usando ontologías en la World Wide Web. Ha sido desarrollado por la W3C. Su objetivo principal es facilitar un modelo de marcado construido sobre RDF y codificado en XML [33] para, finalmente, poder representar ontologías a partir de un vocabulario más amplio y una sintaxis más fuerte que la de RDF. A través de OWL se puede representar de forma explícita el significado de términos pertenecientes a un vocabulario y definir las relaciones existentes entre ellos.

OWL hace posible, junto al entorno RDF el proyecto Web Semántica. Este lenguaje utiliza su conexión con RDF para añadir a las ontologías capacidades como:

- Capacidad para ser distribuidas a través de varios sistemas.
- Capacidad para ser escalables a las necesidades de la Web.

[34] El lenguaje OWL da bastantes más facilidades para expresar significado y semántica de la que pueden dar XML, RDF o RDFS, por ese motivo es el lenguaje más usado en para representar ontologías en la Web, pues añade más vocabulario para describir propiedades y clases: relaciones entre clases, cardinalidad, igualdad, características de propiedades, clases enumeradas, ... (Fig 2-21)

```
...
<owl:Ontology rdf:about="">
  <dc:creator rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Michael Jackson</dc:creator>
  <owl:imports rdf:resource="http://protege.stanford.edu/plugins/owl/dc/protege-dc.owl"/>
  <dc:creator rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Marisol</dc:creator>
  <dc:title rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >ACGT Master Ontology</dc:title>
  <owl:imports rdf:resource="http://www.ifomis.org/obo/ro/1.0"/>
  <dc:publisher rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Insitute for Formal Ontology and Medical Information Science (IFOMIS)</dc:publisher>
  <dc:creator rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Pepa Flores</dc:creator>
  <dc:creator rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Pepe Cocos</dc:creator>
  <owl:imports rdf:resource="http://www.ifomis.org/bfo/1.1"/>
 </owl:Ontology>
 <owl:Class rdf:ID="TreatmentCycle">
  <rdfs:subClassOf>
   <owl:Class rdf:ID="Cycle"/>
  </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:ID="Herceptin">
  <rdfs:subClassOf>
   <owl:Class rdf:ID="PharmaceuticalSubstance"/>
  </rdfs:subClassOf>
  <owl:disjointWith>
   <owl:Class rdf:ID="Tamoxifen"/>
  </owl:disjointWith>
  <owl:disjointWith>
   <owl:Class rdf:ID="Lomustine"/>
  </owl:disjointWith>
  <owl:disjointWith>
   <owl:Class rdf:ID="Vincristine"/>
  </owl:disjointWith>
  <owl:disjointWith>
   <owl:Class rdf:ID="Mannitol"/>
...
```

Fig 2-21 Ejemplo lenguaje OWL

2.2.6 Bases de datos

Se entiende como base de datos a un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso [35]. También podemos decir que es un conjunto de información almacenada en memoria junto a un conjunto de

programas que manipulan estos datos. [36] Estos datos están estructurados y organizados independientemente de su utilización y su implementación.

Surgen a mediados de los sesenta; en 1970 Codd propuso el modelo relacional, que ha marcado la línea de investigación durante muchos años; hoy en día podemos encontrar los modelos orientados a objetos.

Existen distintos modelos de bases de datos:

- Bases de datos jerárquicas: almacenan la información en una estructura jerárquica, enlazando los registros en forma de estructura de árbol, de forma que un nodo padre de información puede tener varios nodos hijo. Este tipo de base de datos es relativamente limitada, pues sólo contiene ligaduras simple (las de tipo 1:1 o 1:N del modelo relacional) y son incapaces de representar la redundancia de datos. Sin embargo son muy útiles en el caso de aplicaciones que manejen un gran volumen de información y datos muy compartidos.
- Bases de datos de red: similares a las bases de datos jerárquicas, pero en este caso se permite que un nodo tenga varios padres (por tanto acepta relaciones de tipo M:N). Con este tipo de modelo se ofreció una solución al problema de la redundancia de datos.
- Bases de datos relacionales: tienen como idea fundamental el uso de las relaciones. A diferencia de otros modelos como el jerárquico o el de red, no tiene relevancia el lugar y la forma de almacenaje de los datos. La información se puede almacenar o recuperar a través de consultas que ofrecen una amplia flexibilidad.
- Bases de datos orientadas a objetos: se usan fundamentalmente en los modelos informáticos orientados a objetos, y trata de almacenar en la base de datos los objetos al completo (estado y comportamiento)
- Bases de datos documentales: permiten la indexación a texto completo siendo posible realizar búsquedas más potentes.
- Bases de datos deductivas: en este tipo de sistema de base de datos se permite hacer deducciones a partir de inferencias. Principalmente está basado en reglas y hechos que se almacenan en la base de datos.

- Bases de datos distribuidas: surgen por la existencia física de organismos descentralizados. Los datos se encuentran almacenados en distintas computadoras conectadas a la red.

Encontramos ciertas ventajas en el uso de bases de datos [37]:

- Independencia de los datos y su tratamiento: un cambio en los datos no implica un cambio en los programas que los tratan.
- Mayor eficiencia en la gestión de almacenamiento.
- Control sobre la redundancia de datos: los sistemas de ficheros almacenan varias copias distintas de los mismos datos en ficheros distintos, desaprovechando espacio de almacenamiento, además de provocar la falta de consistencia de datos. En las bases de datos esos ficheros se integran de forma que no se almacenan varias copias de los mismos, si bien no se puede eliminar la redundancia completamente, pues en ocasiones es necesaria para modelar las relaciones entre los datos.
- Consistencia de datos: al eliminar o controlar las redundancias de datos se reduce en gran medida el riesgo de que haya inconsistencias.
- Más información sobre la misma cantidad de datos: al estar los datos integrados, se puede obtener información adicional sobre los mismos.
- Compartición de datos: a diferencia de los ficheros, que pertenecen a personas o a departamentos que los utilizan, una base de datos pertenece a una empresa y puede ser compartida por todos los usuarios que estén autorizados.

A continuación se detalla el modelo relacional, ya que, aparte de ser el escogido para el desarrollo de este Trabajo de Fin de Carrera, es el más usado en la actualidad para modelar problemas reales y administrar información dinámicamente

2.2.6.1 Bases de datos relacionales

El modelo relacional se basa en el almacenamiento de los datos y de las relaciones existentes entre ellos. Los datos se estructuran a nivel lógico, a modo de tablas formadas por filas y columnas, aunque a nivel físico pueden tener una estructura completamente distinta.

Sus fundamentos fueron postulados en 1970 por Edgar Frank Codd [38], de los laboratorios IBM en San José (California); y se consolidó poco tiempo después como un nuevo paradigma en los modelos de base de datos.

El modelo relacional tiene como estructura fundamental la relación. Esta relación es una tabla constituida por filas (tuplas), que corresponden a registros individuales, y columnas, que corresponden a los campos o atributos de esos registros. El lugar y la forma en que se almacenen los datos no tienen relevancia. Esto supone una ventaja, pues lo hace más fácil de entender y utilizar por un usuario casual de la base de datos. La información se puede recuperar o almacenar por medio de consultas que ofrecen una amplia flexibilidad y capacidad para administrar la información

El modelo relacional es, con diferencia, el modelo lógico en el que se basan la mayoría de los Sistemas Gestores de Bases de Datos (SGBD) hoy en día.

2.2.6.2 Sistemas de Gestión de Bases de Datos (SGBD)

Son un tipo de software específico, dedicado a servir de interfaz entre bases de datos, usuarios y aplicaciones que utilicen las bases de datos (Fig 2-22). Un SGBD puede ser accedido desde un programa, permitiéndose de esta manera la gestión y consulta de las bases de datos ofrecidas por el sistema.

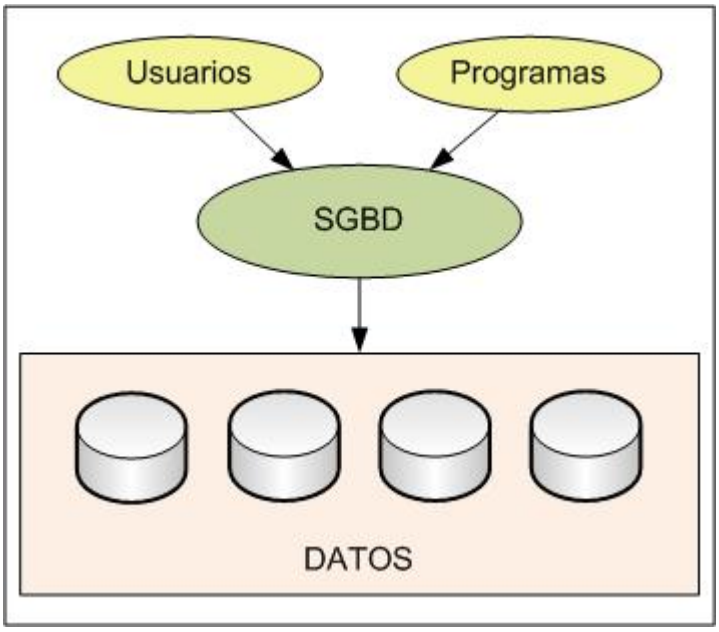

Fig 2-22 SGBD

Un sistema de gestión de bases de datos permite a los usuarios crear, definir y mantener la base de datos, proporcionando acceso controlado a la misma. Está compuesto por dos lenguajes:

- Un lenguaje de definición de datos que permita especificar la estructura que almacenará los datos en la base de datos, los tipos de datos y sus restricciones.
- Un lenguaje de manipulación de datos para insertar, actualizar, eliminar o consultar los datos almacenados en la base de datos. El lenguaje de manipulación de datos más usado es SQL (*Structured Query Language*) [39]. SQL es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar distintos tipos de operaciones sobre éstas [40].

Debido a la existencia de los sistemas de gestión de bases de datos, conseguimos varias ventajas:

- Integridad de los datos: refiriéndose a la validez y la consistencia de los datos almacenados. El SGBD se encarga de mantener las restricciones o reglas que expresan la integridad.
- Seguridad: es la protección de la base de datos frente a usuarios no autorizados. Las autorizaciones pueden darse también a nivel de operaciones, de forma que un usuario podría consultar ciertos datos pero no actualizarlos.

- Accesibilidad a los datos: la gran mayoría de los SGBD proporcionan lenguajes de consulta que permiten al usuario hacer cualquier tipo de consulta sobre los datos.
- Productividad: el SGBD proporciona muchas de las funciones estándar que el programador necesita, por ejemplo las rutinas de manejo de ficheros típicas. De esta forma, el programador puede centrarse mejor en la función específica requerida por los usuarios, sin preocuparse de los detalles de implementación de bajo nivel.
- Independencia de datos: los SGBD separan las descripciones de los datos de las aplicaciones, lo cual simplifica el mantenimiento de las aplicaciones que acceden a la base de datos, que no se ven afectados por un cambio en la estructura de las mismas o en el modo en que se almacenen en disco.
- Concurrencia: los SGBD permiten el acceso compartido a las bases de datos.
- Servicios de copias de seguridad y de recuperación ante fallos: restablecen la base de datos tras un fallo hardware o software.

Existe un SGBD relacional, multihilo y multiusuario, se trata de MySQL. Funciona en múltiples plataformas. MySQL es software open source. Es una base de datos muy rápida en la lectura cuando utiliza un motor no transaccional MyISAM, pero, sin embargo, puede causar problemas de integridad en entornos de alta concurrencia en modificación de datos. En las aplicaciones Web hay baja concurrencia en modificación de datos, pero en la lectura, la concurrencia es muy elevada; esto hace ideal el uso de MySQL en este tipo de aplicaciones.

2.2.7 Servicios Web

Existen múltiples definiciones sobre lo que podría ser un Servicio Web. Una de las más adecuadas sería hablar de ellos como un conjunto de aplicaciones o tecnologías con capacidad para interoperar en la Web [41]. Los Servicios Webs tienen el fin de ofrecer unos servicios intercambiando datos entre dichas aplicaciones o tecnologías.

Un Servicio Web está definido por una interfaz, a través de la cual se tiene acceso a él, y por una URI (*Uniform Resource Identifier*). Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios solicitan un servicio a través de la Web [42].

En la Fig 2-23 se puede ver un ejemplo de interacción entre distintos Servicios Web. Los distintos servicios interoperan entre ellos, mediante protocolos WSDL y SOAP (que se tratarán posteriormente), para encontrar el viaje que desea el cliente.

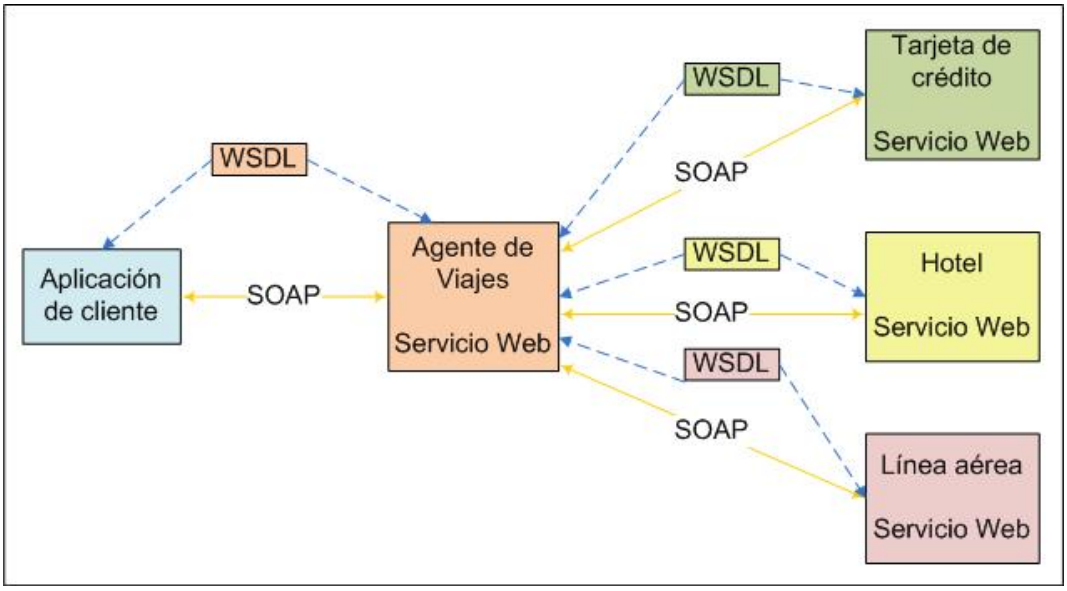

Fig 2-23 Ejemplo de interacción de un conjunto de Servicio Web

En todo este proceso de circulación de información intervienen las tecnologías:

- SOAP (*Simple Object Access Protocol*): recomendación del W3C. Define el formato de los mensajes intercambiados por los servicios Web; permite que varios dispositivos se comuniquen entre sí. Este protocolo está basado en XML y con él se pueden transmitir mensajes a través de HTTP, SMTP, etc. SOAP se encarga de especificar el formato que deberán seguir estos mensajes (Fig 2-24). El formato de estos mensajes será:
  - *Envelope*: para especificar datos globales, como la codificación del contenido, espacios de nombres, etc.
  - *Header*: es un campo opcional. Contiene metainformación sobre el mensaje.
  - *Body*: contiene los datos en formato XML.
- WSDL (*Web Services Description Language*): recomendación del W3C. Describe la interfaz pública del Servicio Web. En el documento WSDL de un Servicio Web, un cliente puede determinar qué se puede hacer con el servicio,

dónde se encuentra, qué parámetros maneja o cómo se invoca. Está estructurado en capas.

- UDDI (*Universal Description, Discovery and Integration*): desarrollado por el consorcio formado por IBM, SUN, Microsoft, Oracle, etc. Tiene dos componentes principales: descripción de negocios y registro de servicios.

Los Servicios Web son ventajosos porque [43]:

- Aportan interoperabilidad entre aplicaciones software, independientemente de cuáles sean sus propiedades o de cuáles sean las plataformas sobre las que se instalen.
- Al usarse en la Web, fomentan los estándares y protocolos basados en texto, por lo que es más fácil acceder a su contenido y entender su funcionamiento.

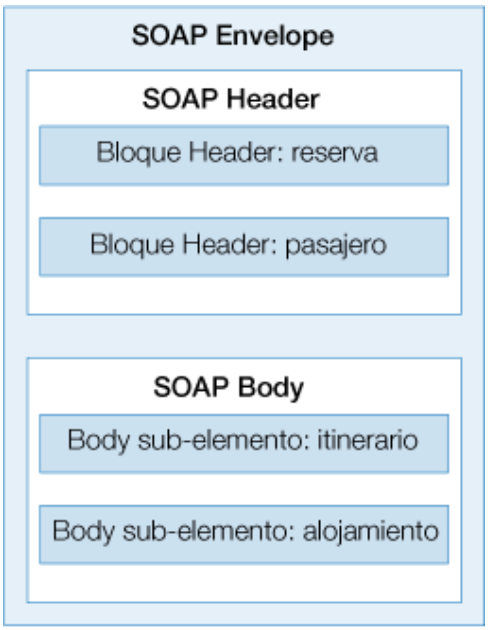

Fig 2-24 Estructura de un mensaje SOAP

- Pueden aprovecharse de los sistemas de seguridad *firewall* sin necesidad de cambiar las reglas de filtrado. Esto se debe a que se apoyan en HTTP.
- Son aplicaciones autocontenidas y modulares.

- La carga de ejecución se traspasa del cliente al servidor (puesto que el servicio se ejecuta en el servidor). De esta forma, el cliente no depende de los recursos de su máquina.
- Pueden proveer servicios integrados, permitiendo que servicios y software de distintas compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente.
- Las actualizaciones del Servicio Web son transparentes al cliente.
- Permiten la interoperabilidad entre distintas plataformas por medio de protocolos estándar abiertos. Las especificaciones las gestiona la organización W3C, por lo que se garantiza que no haya secretismos por intereses particulares de fabricantes concretos y se garantiza la interoperabilidad plena entre aplicaciones.

La principal limitación de los Servicios Web es que necesitan de una conexión a Internet para poder ser usados. Por esta razón tampoco tendría mucho sentido usarlos si los parámetros de la invocación al Servicio Web o su resultado tienen un tamaño considerable, como puede ser un archivo de vídeo.

Para invocar a un Servicio Web se dan los siguientes pasos (Fig 2-25):

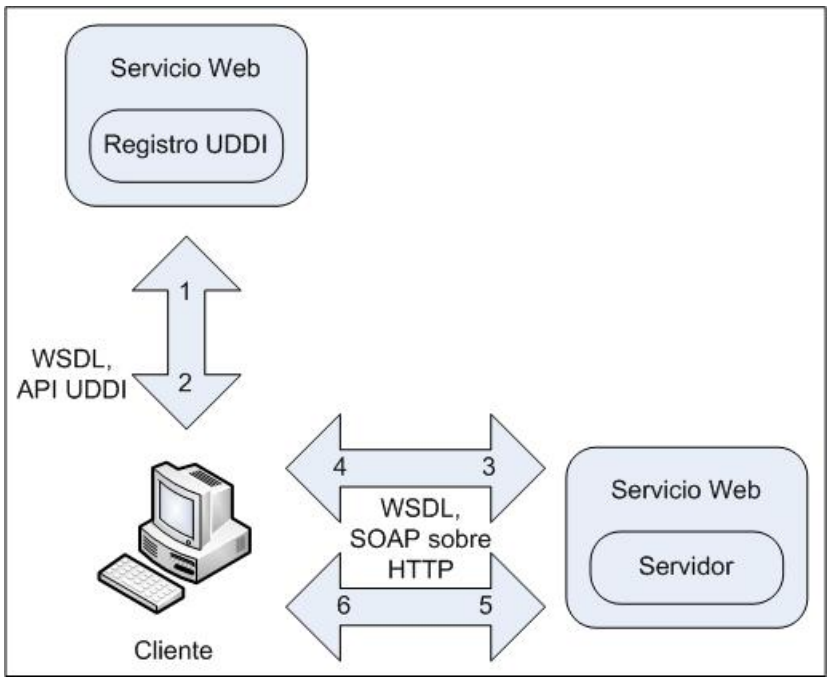

Fig 2-25 Invocación de un Servicio Web

1. El cliente debe localizar el servicio Web usando un registro UDDI (*Universal Description, Discovery and Integration*) o cualquier otro sistema que permita revisar las descripciones WSDL de los servicios Web.
2. El registro UDDI responderá devolviendo una dirección URI.
3. El cliente pregunta al servidor cómo invocar el servicio, puesto que ahora la ubicación mediante la URI, pero no cómo invocarlo.
4. El servidor Web responde con un fichero WSDL donde describe la interfaz del servicio Web a invocar.
5. Se realiza la invocación, normalmente sobre SOAP.
6. El servicio Web contesta con la respuesta SOAP en un mensaje XML.

2.2.8 GRID

Es una tecnología innovadora (desarrollada en ámbitos científicos por Ian Foster y Carl Kesselman a principios de los años 90) que permite utilizar de forma coordinada todo tipo de recursos que no están sujetos a un control centralizado. En este sentido es una nueva forma de computación distribuida, en la cual los recursos pueden ser heterogéneos (ordenadores, estaciones de trabajo, PDA, móviles, diferentes arquitecturas, clústeres, etc) y se encuentran conectados mediante redes de área extensa como Internet [44].

“Un Grid es un sistema que coordina recursos, que no están sujetos a un control centralizado, usando interfaces y protocolos estándares, abiertos y de propósito general para proveer de servicios relevantes”[44].

En ocasiones, un único nodo no puede realizar un trabajo él solo, por motivos de potencia de cálculo o simplemente por no ser capaz de almacenar toda la información asociada a un recurso. Sin embargo, la potencia combinada de varias asociaciones o nodos sí pueden hacerlo. Ésta es la solución que propone la computación GRID: conseguir una mayor potencia de cálculo (prácticamente ilimitada), almacenamiento, aprovechamiento de recursos, etc, combinando los recursos computacionales de varias organizaciones [45][46]. Es una solución altamente escalable, potente y flexible, con la que se evitarán problemas de falta de recursos (cuellos de botella) y que nunca se quedará obsoleta, por la posibilidad de modificar el número y características de sus componentes.

Como se muestra en la Fig 2-26, en la computación “tradicional”, una organización debe hacerse cargo de todas sus necesidades computacionales utilizando sus propios recursos. En la computación GRID siempre intervienen varias organizaciones, cada una con sus propios recursos computacionales. En este tipo de computación, los recursos se agrupan dinámicamente, formando *organizaciones virtuales*, para resolver problemas concretos.

Las *organizaciones virtuales* son un concepto fundamental dentro de la computación Grid. Consisten en una agrupación de recursos de varios individuos y/o organizaciones distintas que colaboran para alcanzar una meta común. Una característica importante es que la pertenencia a una *organización virtual* no es permanente.

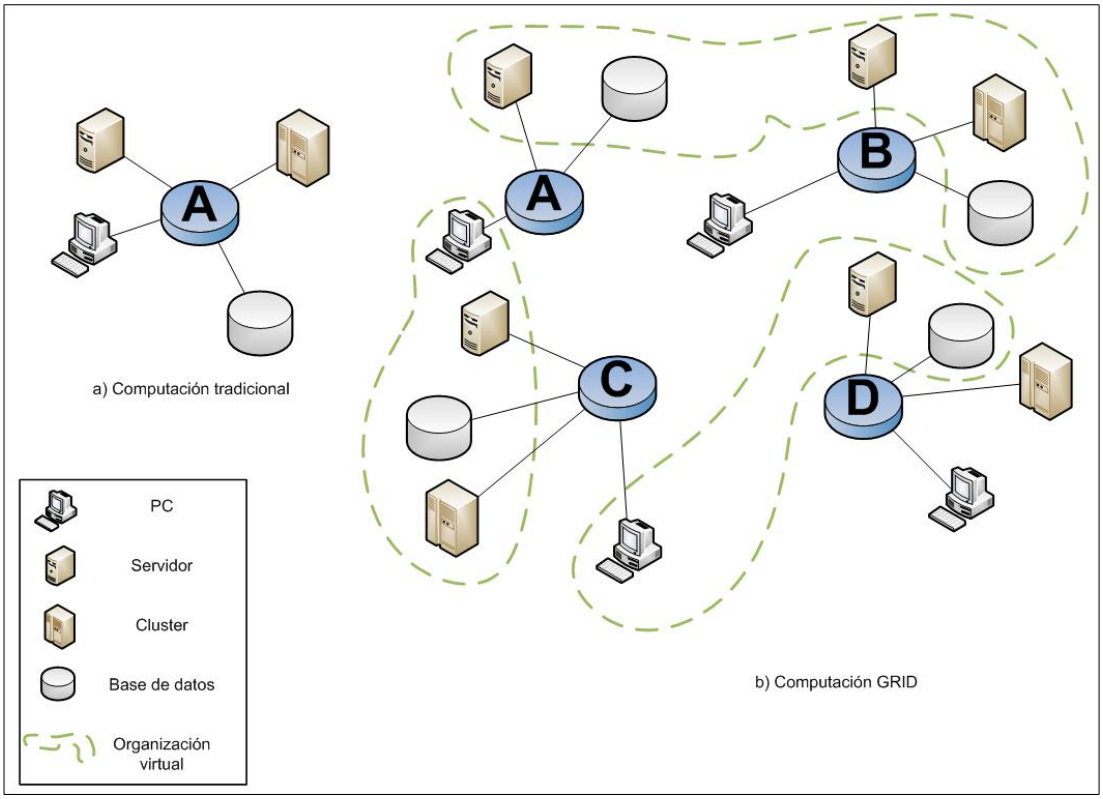

Fig 2-26 Computación tradicional vs computación GRID

En la actualidad existen muchos proyectos desarrollados en esta línea (realización de trabajos en distintos nodos), como Emule, Edonkey o Limewire, que son programas de compartición de datos entre diferentes máquinas a nivel mundial. Las tecnologías Grid y *Peer-to-peer* (P2P) tienen bastante en común, partiendo de la idea de compartición de recursos; si bien, se puede ver la P2P como más anónima y generalizada en ordenadores

de Internet, mientras que la tecnología Grid nace de una estructura de nodos más controlada y jerarquizada en centros científicos [44].

Otro proyecto bastante conocido es el Seti@Home, que cuenta con miles de computadores repartidos por Internet que ceden tiempo de sus procesadores, ciclos de proceso desocupados, para analizar señales buscando patrones inteligentes extraterrestres.

Las características de la arquitectura GRID serían las siguientes:

- Heterogeneidad a gran escala.
- Compartición y coordinación de recursos.
- Múltiples dominios de administración.
- Seguridad de acceso.
- Eliminación de las distancias en la distribución geográfica.
- Uso de estándares abiertos.
- Acceso transparente, fiable, consistente y pervasivo.

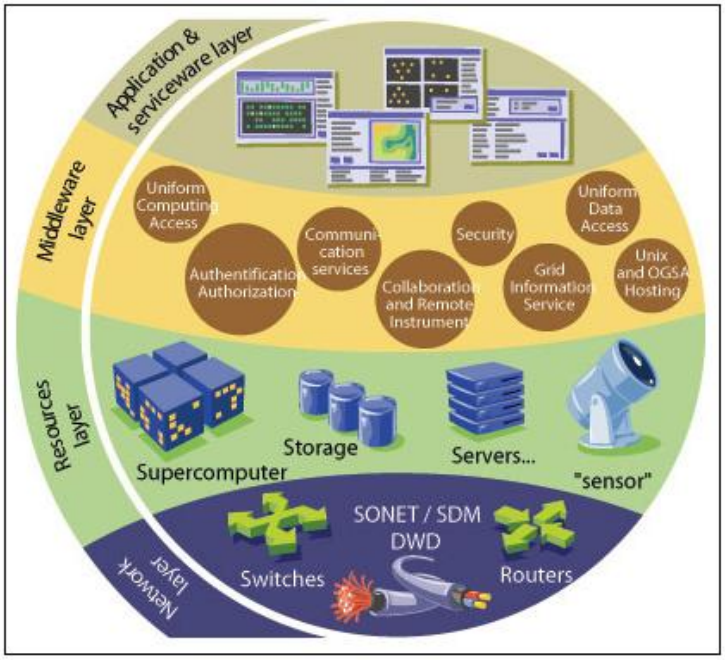

Fig 2-27 Arquitectura Grid (desde http://www.gridcafe.org)

La arquitectura Grid es una arquitectura basada en capas (Fig 2-27):

- Capa de Red: formada por dispositivos físicos (cables, tarjetas, switches, routers, etc) y protocolos de comunicaciones a bajo nivel que sirven para permitir la interconexión de los recursos.
- Capa de Recursos: formada por los recursos integrados y compartidos dentro del Grid, como pueden ser CPUs, servidores, dispositivos de almacenamiento, instrumentos, sensores, etc.
- Capa de middleware: es el “cerebro del Grid”, funciona de intermediario entre las aplicaciones y los recursos. Tiene diferentes servicios de: acceso, autenticación, seguridad, información, monitorización, etc.
- Capa de aplicaciones: formada por herramientas y/o aplicaciones de los usuarios del Grid. En esta capa se utilizan los recursos ofrecidos por el Grid a través de los servicios ofrecidos por la capa de middleware. Las aplicaciones pueden ser de distintos tipos: cálculo, visualización, simulación, monitorización, etc.

Existen diferentes tipos de Grid, dependiendo de las aplicaciones:

- Grid computacional fue el primero que surgió. Consiste en un conjunto grande de recursos computacionales que se encuentran interconectados para ejecutar distintos trabajos. Estos trabajos son divididos, planificados y distribuidos entre los distintos recursos de forma que sea transparente para el usuario. Básicamente consiste en compartir recursos de cálculo a gran escala.
- Grid de recursos: extiende los recursos a compartir a los datos. Son colecciones de datos y recursos de almacenamiento (grandes bases de datos distribuidas), con propiedades de replicación, tolerancia a fallos y seguridad de acceso. Desde la perspectiva del cliente, este tipo de Grid se usa como un único recurso.
- Grid de aplicación: es un híbrido de los dos anteriores. Este tipo hace uso de conceptos tanto de los Grids computacionales como de los Grids de recursos. Un ejemplo podría ser un servidor de base de datos, en el que una petición de información es distribuida por varias máquinas. Usando esta aproximación, se aumentaría la eficiencia global; cada máquina devolvería sus propios registros que deberían ser “mezclados” antes de ser devueltos al cliente.

El Grid ha evolucionado desde sus orígenes. Se distinguen tres generaciones de Sistemas Grid [47]:

- Primera generación: Metacomputadores. Viene definida por la Metacomputación, es decir, conectar varios ordenadores y clústeres en la red para resolver un problema en un tiempo más razonable, ya que hay trabajos que necesitan mucho tiempo para ejecutarse completamente aunque se use un supercomputador o un clúster. La Metacomputación es una supercomputación distribuida. Un ejemplo de esta generación sería el programa Seti@Home. Esta generación tiene una serie de problemas:
    - Es muy dependiente de la máquina.
    - No está orientado a servicios ni a recursos. Se trata de un Grid Computacional muy básico en el que las CPUs ejecutan lo que se les envíe.
    - Uso de protocolos no estándar.
    - Seguridad: tanto los datos como las aplicaciones son mandadas a los recursos.
- Segunda generación: Grid orientado a recursos. Aparece el concepto de middleware y el desarrollo de protocolos basados en el acceso a recursos (enmascaramiento de heterogeneidades locales, negociación de las políticas de seguridad, integración de múltiples recursos distribuidos y establecimiento de un acceso homogéneo a recursos de distinta naturaleza). En esta generación aparece *Globus Toolkit*. Esta generación cuenta con una serie de problemas también:
    - Los programas cliente siguen "moviéndose" hasta los recursos.
    - No es orientado a servicios.
    - Demasiado orientado al sistema operativo *Linux*.
- Tercera generación: Grid orientado a servicios. Es el resultado de la convergencia entre el Grid y los servicios Web. En esta generación se incorporan la arquitectura OGSA y la especificación WSRF (Fig 2-28).

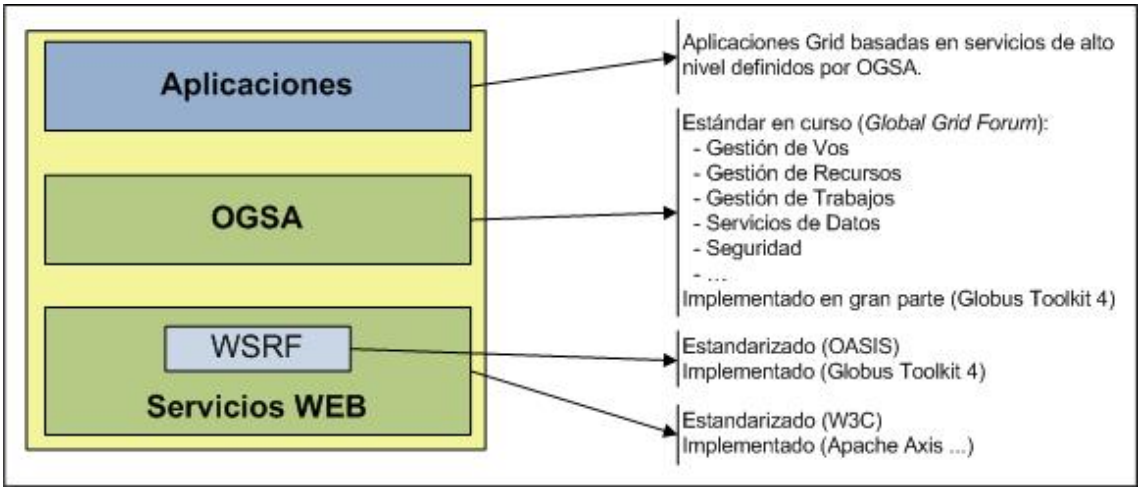

Fig 2-28 Esquema de la tercera generación Grid

Por sus características, las principales áreas de aplicación de este tipo de computación son:

- Aplicaciones con grandes necesidades computacionales: como simulación, predicción o monitorización.
- Aplicaciones con grandes necesidades de almacenamiento o proceso de datos: aplicaciones que generan un flujo grande y constante de datos o que se benefician del acceso a datos similares situados en distintas asociaciones.
- Aplicaciones colaborativas: aplicaciones que, por su naturaleza, son multi-organización y se benefician de una tecnología que facilita la comunicación entre distintas organizaciones.

2.2.8.1 OGSA y OGSI

La OGSA (*Open Grid Services Architecture*) es la arquitectura Grid definida por la organización *Global Grid Forum* (consorcio de más peso dentro de la definición de estándares en Computación Grid). Consiste en una serie de instrucciones sobre cómo construir un servicio Grid, con qué componentes y cómo ensamblarlos.

El *Global Grid Forum* (GGF) define OGSA como [48]:

“Es un marco (framework) de amplio espectro de aplicación para la integración de Sistemas Distribuidos”. La plataforma OGSA se constituye por dicho marco, el cual define un núcleo de interfaces, comportamientos, modelos de recursos, enlaces, etc. El

grupo de trabajo OGSA, dentro del GGF, define esta plataforma mediante una serie de pasos [48]:

a) Definición del alcance de los servicios que se necesitan para dar un soporte adecuado para las aplicaciones de Grid (e-science, e-business, etc).
b) Identificación de un subconjunto básico de dichos servicios que sean esenciales para los sistemas Grid y sus aplicaciones. Definir un conjunto base (*core*) de servicios.
c) Especificación de las funcionalidades de alto nivel de los servicios base y sus interrelaciones.

La arquitectura definida por OGSA se encuentra implementada en la plataforma OGSI (*Open Grid Services Infrastructure*).

La definición que da el GGF de OGSI es[49]:

"OGSI define mecanismos para crear gestionar e intercambiar información entre entidades llamadas Servicios Grid utilizando tecnología de Grid y Servicios Web. Sucintamente, un servicio Grid es un Servicio Web que se ajusta a una serie de convenciones (interfaces y comportamientos) que definen como los clientes interaccionan con el Servicio Grid. Estas convenciones y otros mecanismos OGSI asociados con la creación y localización (*discovery*) de servicios Grid permiten una gestión adecuada (control, fiabilidad, seguridad) de la información de estado distribuido y de larga duración que se requiere en la ejecución de aplicaciones distribuidas."

2.2.8.2 Servicios Grid

Un Servicio Grid es una ampliación de los servicios Web y su arquitectura fue especificada por el *Global Grid Forum* (OGSA). Surgieron de la idea de buscar una tecnología que se adaptara a las necesidades de una aplicación Grid. Los servicios Web no fueron suficiente para esta idea puesto que presentaban algunas limitaciones, que se superaron en los servicios Grid:

- Los servicios Web no mantienen el estado de una invocación a otra, mientras que los servicios Grid sí lo hacen.

- Los servicios web no son transientes, no se pueden crear varias instancias de un mismo servicio y destruirlas según se necesiten o ya no sean necesarias. En los servicios Grid sí se puede.
- Los servicios Web no incluyen servicios de apoyo como las notificaciones, el servicio de persistencia, la gestión del ciclo de vida, etc.

Los servicios Grid utilizan un enfoque de Factorías de Objetos. Los servicios Web tienen un único servicio compartido por todos los usuarios, pero en este otro tipo de servicios se tiene una especie de "servicio-factoría" que crea instancias individuales de los servicios, de forma que cuando se invoca a una operación del servicio se directamente a dicha instancia. No sólo esto es posible, también es posible crear una instancia, o varias por cliente, o una para varios clientes.

Los servicios Grid son Servicios Web pero con la incorporación de la plataforma OGSI. Los mecanismos incorporados son:

- Servicio de nombres (*Naming*) para asegurar la existencia de un nombre único para cada instancia del Servicio Grid y para permitir la localización de los servicios (*discovering*)
- Servicio de datos para gestionar los datos asociados a la ejecución del servicio.
- Notificación: son los mecanismos utilizados para la comunicación entre los componentes de una aplicación Grid; están formados por el conjunto de interfaces para registrar y suministrar notificaciones y subscripciones.
- Ciclo de Vida: son los mecanismos para la persistencia, creación y destrucción de instancias de los Servicios Grid.

2.2.8.3 WSRF

WSRF (*Web Service Resource Framework*) forma parte de la familia de especificaciones sobre servicios web publicadas por OASIS [50].

Un Servicio Web no mantiene ninguna información entre sucesivas invocaciones y esto limita el número de cosas que se pueden hacer con ellos, aunque existan soluciones. WSRF es el resultado de la convergencia entre las tecnologías Web y Grid.

WSRF proporciona un conjunto de operaciones para que los servicios web puedan convertirse en Servicios Web con estado. Los clientes de los servicios se comunican con

servicios WSRF que representan a los recursos y que permiten almacenar y recuperar la información. De esta forma, los clientes invocan al servicio añadiendo como parámetro el identificador del recurso que se utilizará durante la petición, codificado en una referencia que cumpla *WS-Addressing* (mecanismo por el cual se pueden identificar servicios web; esta referencia puede consistir en una simple URI o hasta en un fichero XML).

También proporciona un conjunto de operaciones estandarizadas para la consulta y modificación de las propiedades del recurso.

La relación entre WSRF, la arquitectura Grid, y los Servicios Web se muestra claramente en la siguiente figura (Fig 2-29)

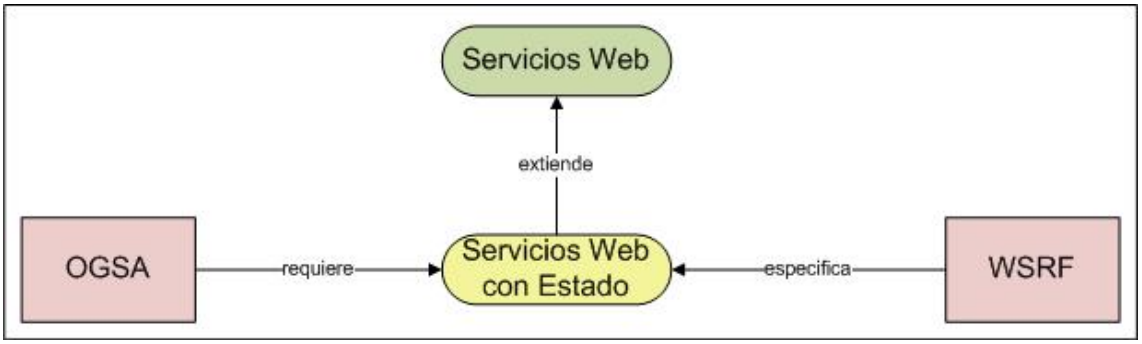

Fig 2-29 Relación WSRF, OGSA y Servicios Web

WSRF es uno de los principios de la especificación WSDM (*Web Services Distributed Management*), dedicada a la gestión de recursos distribuidos.

2.2.8.4 Globus Toolkit

El *Globus Toolkit* (GT) es un conjunto de herramientas software de código abierto utilizadas para construir sistemas y aplicaciones GRID, que está siendo desarrollado por la Globus Alliance [51].Estas herramientas pueden ser usadas juntas o de forma independiente. Proporciona un acceso uniforme y seguro a recursos de almacenamiento, incluye software para seguridad, gestión de ejecución de tareas, de datos y de información. Actualmente, Globus es el estándar *de facto* para la computación Grid [48]. Las claves de su éxito son la orientación a objetos y que es open source.

Proporciona una serie de servicios de alto nivel para la construcción de aplicaciones Grid:

- Monitorización de recursos.
- Descubrimiento de servicios.
- Infraestructura para el envío de trabajos.
- Infraestructura de seguridad.
- Servicios de gestión de datos.

Desde 1998 Globus Toolkit ha sufrido diversas modificaciones:

- GT1 (1998)
- GT2 (2002)
- GT3 (2003)
- GT4 (2005)

El GT1 y GT2 fueron implementados en ANSI C y no usaban ningún tipo de Servicio Web. Desde la versión 2 de *Globus Toolkit*, sus pilares básicos han sido la gestión de recursos y datos, los servicios de información y la seguridad (Fig 2-30).

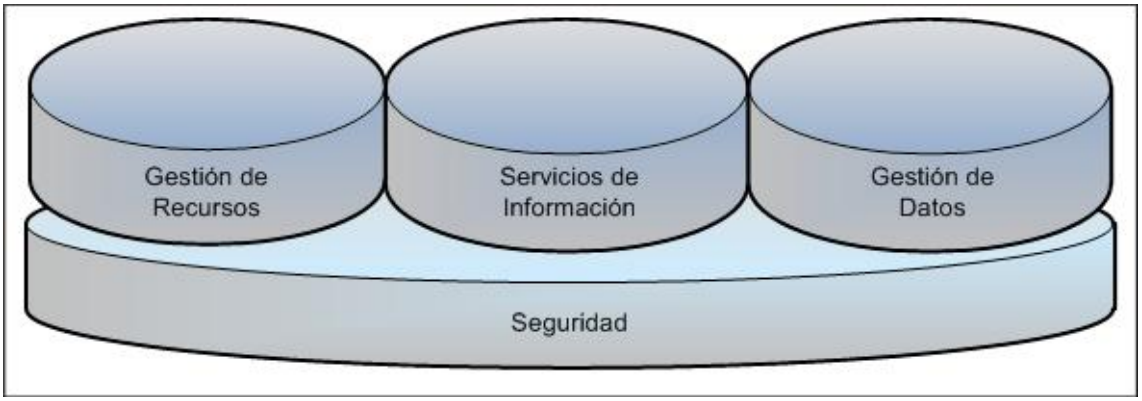

Fig 2-30 Componentes fundamentales de *Globus Toolkit*

En el GT3, ya se cuenta en gran parte con la implementación de OGSA y OGSI. Es el primero que adopta la especificación de los Servicios Web e introduce el concepto de Servicios Grid. Está escrito en Java, por lo que es multiplataforma. Está basado en tecnologías estándar como XML, SOAP, WSDL o Servicios Web.

En el GT4, las características son las mismas que en GT3 a diferencia de que se implementan OGSA y WSRF. La arquitectura de GT4 está dividida en cinco categorías (Fig 2-31) [52]:

- Seguridad: componentes de seguridad basados en *Grid Security Infrastructure* (GSI)
- Gestión de datos: para manejar grandes volúmenes de datos en las organizaciones virtuales.
- Gestión de ejecución: inicialización, monitorización, gestión, planificación y coordinación de trabajos.
- Servicios de información: para monitorización y descubrimiento de servicios.
- Soporte para la ejecución: conjunto de librerías y herramientas fundamentales para la construcción de servicios.

Para implementar esta arquitectura, GT4 tiene una serie de componentes [48]:

- GRAM (*Globus Resource Allocation Manager*): permite un acceso transparente, unificado y seguro a los distintos gestores de recursos locales de cada centro o institución. GRAM procesa peticiones de recursos para ejecución de aplicaciones remotas, maneja los trabajos activos y devuelve la información actualizada de las capacidades y la disponibilidad de los recursos de cómputo.

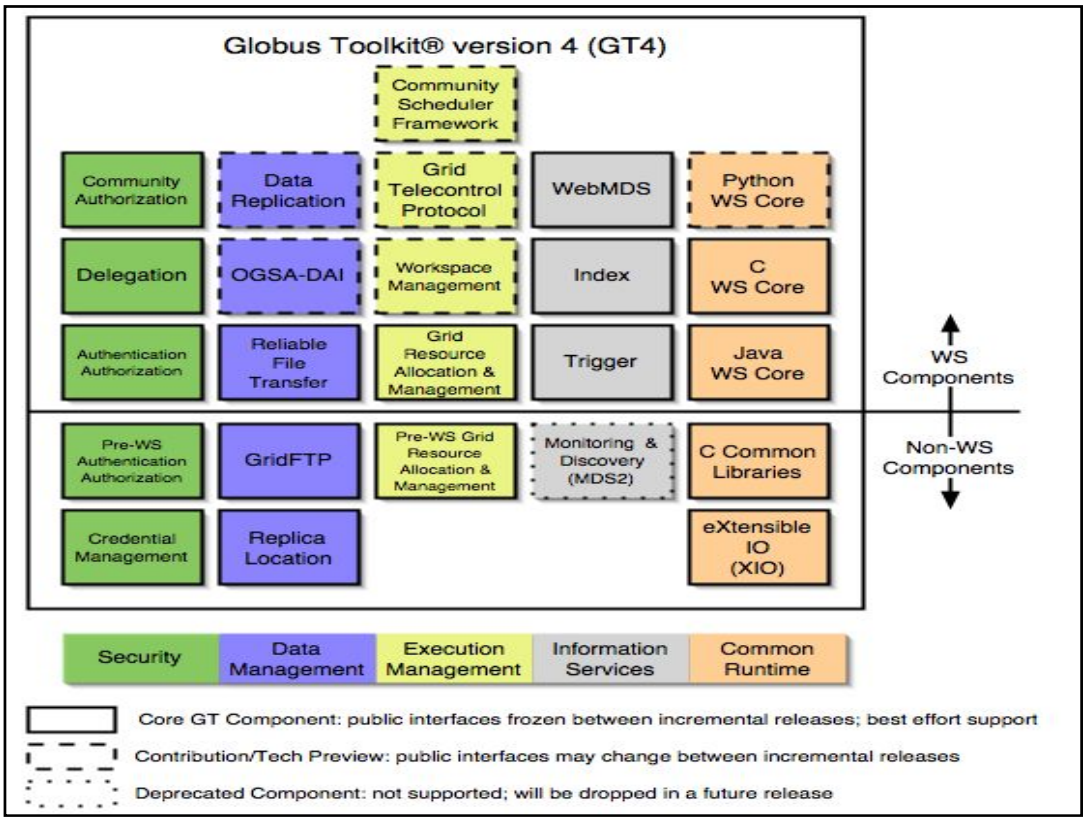

Fig 2-31 Arquitectura de *Globus Toolkit* 4 [52]

- GSI (*Grid Security Infrastructure*): *Globus Toolkit* usa el GSI para establecer una autentificación y una comunicación segura a través de una red abierta como Internet.
- MDS (*Monitoring and Discovery Service*): usa el protocol LDAP (*Lightweight Directory Access Protocol*) para la consulta uniforme de la información asociada a los sistemas en el Grid.
- GRIS (*Grid Resource Information Service*): provee de forma uniforme la búsqueda de recursos obteniendo como resultado las capacidades, el estado, la configuración actual y las prestaciones de cada uno de los recursos Grid.
- GIIS (*Grid Index Information Service*): agrupa la información suministrada por cada GRIS, ofreciendo una imagen conjunta y coherente de los recursos Grid. En definitiva, acepta los mensajes de registro de los GRIS y conjunta dichas fuentes de información en un espacio unificado.
- *GridFTP*: es un protocol de transferencia de ficheros seguro, de alto rendimiento y basado en FTP.
- (GRC) *Globus Replica Catalog*: es un mecanismo para mantener un registro de réplicas de conjuntos de datos.
- (GRM) *Globus Replica Management*: es el mecanismo que une el GRC y el GridFTP para permitir que las aplicaciones creen y administren las réplicas de datos.

2.2.8.5 OGSA-DAI

El objetivo del proyecto OGSA-DAI (*Open Grid Services Architecture Data Access and Integration*) es desarrollar un middleware para asistir en el acceso e integración de datos desde distintas fuentes de datos a través de la Grid [54]. Dicho proyecto está fundado por el programa de eCiencia de UK y por la industria.

El middleware OGSA-DAI proporciona acceso de datos y funciones de integración para los nodos de computación de la Grid que usan OGSA. Sirve para el despliegue de recursos heterogéneos y su acceso vía Web; esos recursos pueden ser bases de datos relacionales, XML, ficheros, etc.

Características de OGSA-DAI:

- Tiene un interfaz único y bien definido.

- Es extensible, por lo que los usuarios pueden añadir sus propias funcionalidades.
- Permite consultar, actualizar y transformar recursos de datos.
- Los servicios Web de OGSA-DAI pueden desplegarse en entornos Grid permitiendo la "gridización" de recursos de datos.
- Es compatible con las especificaciones de servicios Web WS-I (*Web Services Inter-operability*) y WSRF.

OGSA-DAI soporta lo siguiente:

- Diferentes tipos de recursos de datos (incluyendo bases de datos relacionales, XML y archivos) pueden ser expuestos a través de servicios Web.
- Los datos de cualquiera de este tipo de recursos pueden ser consultados y modificados.
- Los datos pueden ser transformados (usando transformaciones XSL), comprimidos y descomprimidos (usando la compresión ZIP y GZIP).
- Los datos se pueden transferir a los clientes, a otros servicios Web de OGSA-DAI, URLs, servidores FTP, servidores GridFTP o a archivos.
- Acceso al listado de recursos ofrecidos y sus funcionalidades.
- Creación de nuevos tipos de recursos (extensibilidad de funcionalidades).

La arquitectura de OGSA-DAI se muestra en la siguiente figura (Fig 2-32)[55]:

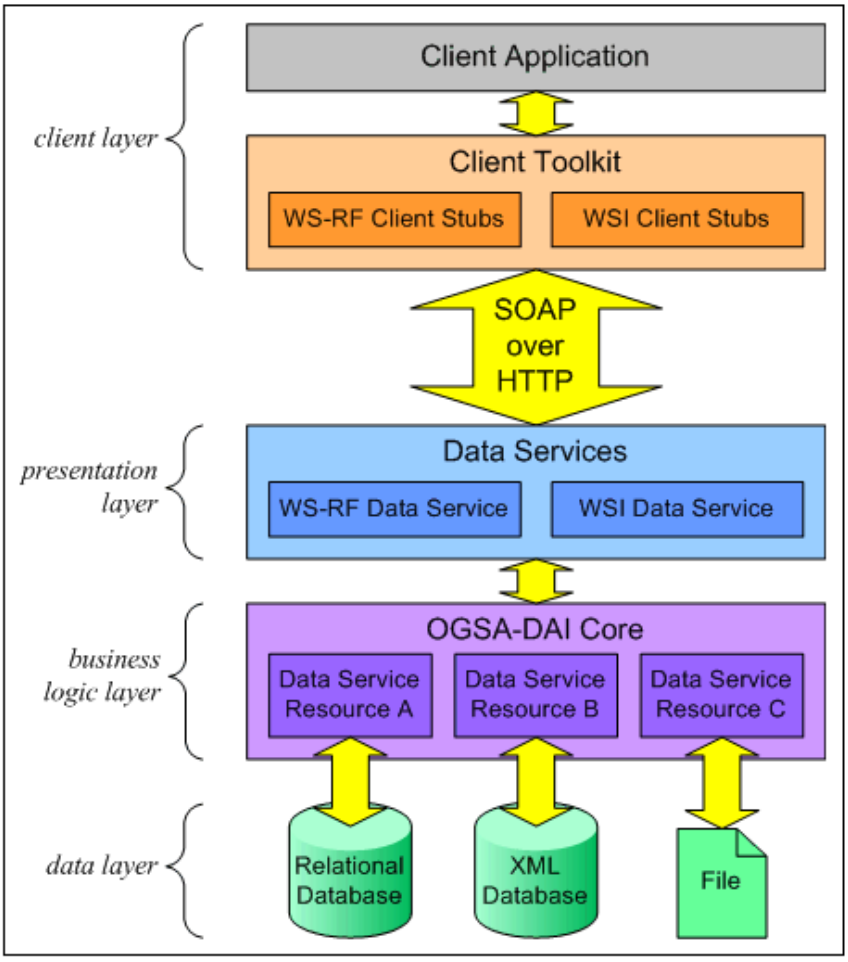

Fig 2-32 Arquitectura OGSA-DAI [55]

La capa de datos (*data layer*) son los recursos de datos que pueden ser expuestos vía OGSA-DAI. Actualmente se incluyen recursos como:

- Bases de datos relacionales como MySQL, SQL Server, DB2, Oracle o PostgreSQL.
- Bases de datos XML como eXist o Xindice.
- Archivos y directorios en distintos formatos (OMIM, SWISSPROT o EMBL).

En la capa de la lógica de negocio (*business logic layer*) se encapsula el núcleo de las funcionalidades de OGSA-DAI. Son una serie de componentes conocidos como *data services resource*. Como se muestra en la figura Fig 2-32 se pueden desplegar múltiples *data services resources* para exponer múltiples recursos de datos. Las responsabilidades de un *data service resource* incluyen:

- Ejecución de los documentos *perform*: un documento *perform* describe las acciones que un *data service resource* debería ejecutar para el cliente. Cada una de dichas acciones se conoce como actividad. OGSA-DAI incluye un gran número de actividades para ejecutar operaciones como consultas a bases de datos, transformaciones de datos y entrega de datos.
- Generación de documentos *response*: un documento *response* contiene el resultado de la ejecución de un documento *perform*.
- Acceso a los recursos de datos.
- Funcionalidad de transporte de datos: los datos resultado pueden ser devueltos o accedidos desde otro *data service resource* o directamente al cliente.
- Control de sesiones.
- Control de propiedades: para el control de las propiedades (creación, acceso o eliminación) asociadas al *data service resource*.

La capa de presentación (*presentation layer*) encapsula las funcionalidades necesarias para exponer los *data service resources* a través de interfaces de servicios Web. Como se mencionó antes, OGSA-DAI incluye dos implementaciones (WSI y WSRF).

La capa cliente (*client layer*) interactúa con el *data service resource* a través del correspondiente *data service* de la capa de presentación.

Por otra parte, son importantes las interfaces definidas entre las capas:

- Capa de datos – capa de la lógica de negocio: la información viaja en los dos sentidos, encargándose de ello el componente conocido como *data resource accessor*. Cada *data service resource* tiene su propio *data resource accessor* que controla el acceso a un recurso de datos subyacente como se muestra en la figura Fig 2-33.

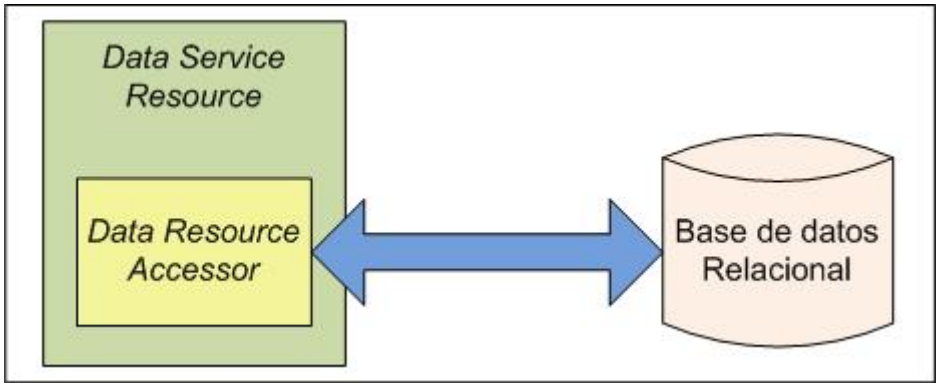

Fig 2-33 Un *data resource accessor* controla el acceso a un recurso de datos

- Capa de la lógica de negocio – capa de presentación: cuando una consulta SOAP llega a los *data services* (WSI o WSRF), esta interfaz se encarga de trasladar la información y las instrucciones hacia la capa de la lógica de negocio. En concreto, el flujo de información entre estas dos capas es el siguiente:
    - Capa de presentación → Capa de la lógica de negocio:
        - Nombres de *data service resources*, nombres de propiedades de *data service resources*, identificadores de sesiones.
        - Certificados y credenciales del proxy del cliente.
        - Documentos *perform* y los datos de los clientes.
        - Información sobre la configuración del *data service resource*, incluyendo los drivers, el usuario y contraseña de la bases de datos, las URIs de los recursos de datos, etc.
    - Capa de la lógica de negocio → Capa de presentación:
        - Documentos response y los datos resultado.
        - El estado del procesamiento de la consulta.
        - Las propiedades del *data service resource*.
        - Información sobre las actividades que soporta el *data service resource*.

Resumiendo, OGSA-DAI soporta la interacción con los *data service resources* a través de interfaces orientados-a-documentos (Fig 2-34).

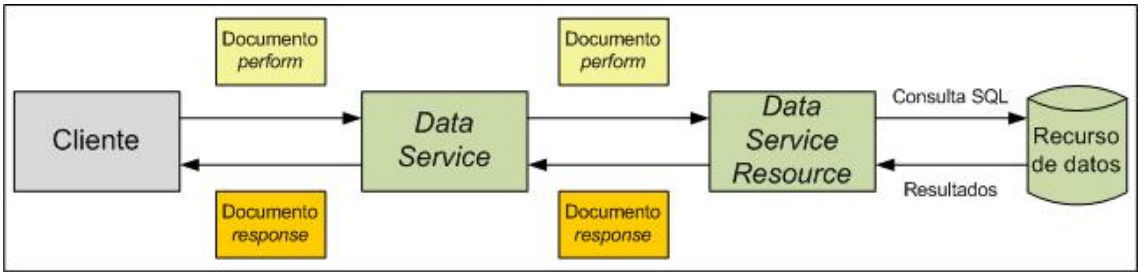

Fig 2-34 Interacción interfaces OGSA-DAI

Es importante conocer las diferencias entre (Fig 2-35):

- *Data service*: es el punto de entrada de los clientes a los recursos expuestos mediante OGSA-DAI. Un *data service* expone cero o más *data service resources* (un *data service* y sus *data service resources* deben residir en el mismo servidor)
- *Data service resource*: ofrece acceso a un único recurso de datos expuesto en OGSA-DAI. Se trata de una relación 1:1. Tanto el *data service resource* como el recurso de datos deben residir ambos en el mismo servidor.
- *Data resource accessors*: permiten que OGSA-DAI se comunique con los recursos físicos. Controla la interacción con el recurso de datos a través del *data service resource.*

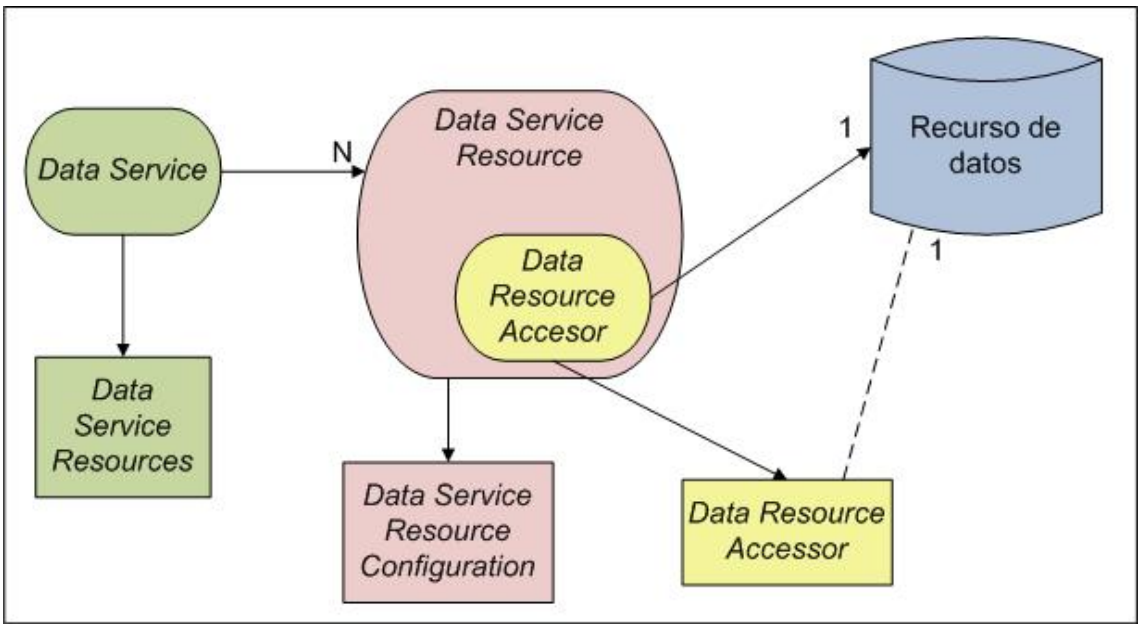

Fig 2-35 *Data services, data service resources* y recursos de datos

Tanto el *data service resource* como el *data resource accessor* tienen asociados una serie de archivos de configuración que especifican las actividades que soportan, las sesiones, el nombre de la clase que lo implementa, etc.

2.2.9 Ontologías

El término ontología proviene de la filosofía griega, y es una disciplina que proviene o pudiera tratarse como una rama de la metafísica general. También es llamada “teoría del ser”, o el estudio de todo lo que es: qué es, cómo es y cómo es posible. Esta disciplina describe las concepciones de la realidad, sus relaciones y características [56].

En informática, tomando el nombre por analogía, una ontología trata de formular un riguroso y exhaustivo esquema conceptual dentro de un dominio dado, para facilitar la comunicación y la compartición de información entre distintos sistemas. Comúnmente, las ramas de la informática donde suelen usarse las ontologías son la inteligencia artificial y la representación del conocimiento. Así, los tipos de programas en los que se utilizan son los dedicados al razonamiento inductivo, la clasificación y una gran variedad de técnicas de resolución de problemas.

Normalmente, las ontologías en los ordenadores están muy relacionadas con vocabularios fijos, en cuyos términos se describe todo lo demás.

Gruber (1992) describe el término ontología como una especificación explícita de una conceptualización, de forma que, una ontología, trata de proporcionar una estructura y unos contenidos de forma explícita que codifique las reglas implícitas de una parte de la realidad; estas declaraciones explícitas deberían ser independientes del fin y del dominio de la aplicación en que se usarán sus definiciones. En la realidad, dichas ontologías son altamente valoradas comercialmente, por lo que se crea una gran competencia para definirlas. Peter Murray-Rust se quejó diciendo que esta competencia conducía a “una guerra semántica y ontológica debido a la competencia entre estándares”, pues cada uno puede tener su propia idea de “lo que existe”.

El desarrollo de sistemas basados en el conocimiento, se suele realizar en diferentes contextos, puntos de vista y distintas suposiciones acerca de la materia de estudio. Cada uno usa su propio vocabulario, por lo que se pueden tener diferentes conceptos con significados que se solapen. A raíz de esto, se crean problemas de comunicación, por la

falta de entendimiento compartido; esto limita la interoperabilidad (capacidad de un sistema para compartir e intercambiar informaciones y aplicaciones), por lo que el potencial de reutilizar y compartir información se ve muy deteriorado.

Según Sheth (1999) [57], la nueva generación de sistemas de información debe ser capaz de resolver la interoperabilidad semántica, en la que un hecho puede tener más de una descripción, para que sea posible el buen uso de las informaciones disponibles como la llegada de Internet y la computación distribuida.

Para esto una buena solución son las ontologías, ya que con ellas se puede crear un entendimiento compartido, unificando distintos puntos de vista y permitiendo establecer correspondencias y relaciones entre los diferentes dominios de entidades de información.

Desde los 90 se han descrito una serie de lenguajes para definir ontologías, desde los basado en reglas marcos (KIF, Ontolingua, OCML, Loom, FLogic) a los más modernos, que se basan en lenguajes de marcado (RDF, OWL, XOL, SHOE, OIL, DAMN+OIL) [58].

2.2.10 Análisis y Diseño orientado a Objetos mediante UML

El lenguaje UML (*Unified Modeling Language*) o Lenguaje Unificado de Modelado es, en los últimos años, un lenguaje aceptado universalmente para el diseño software. Proporciona un método claro para la modelización, construcción y documentación de los elementos que componen un sistema software orientado a objetos. Este lenguaje consiste en un conjunto de especificaciones de notación orientadas a objetos, cada una de las cuales están compuestas por distintos diagramas que representan diferentes etapas del desarrollo de un proyecto software.

Este lenguaje comenzó a gestarse en 1994 [60], cuando Rumbaugh se unió a la compañía Rational fundada por Booch. Ambos tenían el objetivo de unificar los dos métodos que habían creado: el método Booch y el OMT (Object Modeling Tool). En octubre de 1995 apareció el primer borrador del lenguaje, sin embargo fue necesaria la

adhesión de Jacobson al grupo formado por Booch-Rumbaugh, para añadir nuevas ideas a dicho borrador. Este grupo es conocido como los “tres amigos”. También se abrió este lenguaje para la colaboración de otras empresas que también aportaron sus ideas. Ese conjunto de ideas condujeron a la definición de la primera versión de UML, la cual fue entregada a un grupo de trabajo que, en 1997, lo convirtió en un estándar del OMG (Object Management Group).

El modelado de un sistema es básico en la construcción de software. Su objetivo es capturar las partes esenciales del sistema, realizando una abstracción del mismo y plasmándolo a través de una notación gráfica (lo cual se conoce como modelado visual). El modelado visual permite manejar la complejidad del sistema a analizar o diseñar, proporcionando los distintos planos del sistema, que serán más o menos detallados en función de los elementos que se deseen resaltar en cada momento.

En este sentido, el lenguaje UML permite el modelado visual completo de sistemas complejos. Además es independiente del lenguaje en el cual se realice la implementación, aunque es más utilizable con lenguajes orientados a objetos.

UML aporta una serie de ventajas al modelado:

- Mayor rigor en la especificación.
- Permite realizar una validación y verificación del modelo realizado.
- Permite la automatización de determinados procesos como es la generación de código a partir de los modelos y el proceso inverso. Esta automatización es especialmente útil por permitir que tanto código como modelo estén actualizados el uno respecto al otro, por lo que siempre se puede mantener la visión en el diseño de la estructura de un sistema.

El lenguaje UML tiene una serie de funciones principales:

- Visualización: permite expresar de forma gráfica un sistema, de tal forma que una persona que no conozca nada del mismo pueda entenderlo.
- Especificación: permite especificar las características de un sistema antes de su construcción.
- Construcción: como se ha comentado, es posible construir los sistemas diseñados a partir de los modelos especificados.

- Documentación: los propios elementos gráficos sirven como documentación del sistema a desarrollar y pueden ser útiles para su futura revisión.

2.2.10.1 Notación gráfica en UML

La notación gráfica del lenguaje UML es lo suficientemente expresiva como para permitir la representación de todas las fases en la construcción de un proyecto, desde el análisis con sus casos de uso, el diseño con los diagramas de clases, objetos, paquetes, etc., hasta la implementación y configuración con los diagramas de despliegue.

Un modelo UML se compone de tres tipos de bloques de construcción:

- Elementos: son abstracciones de cosas reales o ficticias.
- Relaciones: sirven para relacionar los elementos entre sí.
- Diagramas: son colecciones de elementos con sus respectivas relaciones. En concreto, ofrece una vista general o particular del sistema a modelar.

UML ofrece una amplia variedad de diagramar para poder representar correctamente un sistema y poder visualizarlo desde distintas perspectivas. En concreto:

- Diagrama de casos de uso
- Diagrama de clases
- Diagrama de paquetes
- Diagrama de objetos
- Diagrama de secuencia
- Diagrama de colaboración
- Diagrama de estados
- Diagrama de actividades
- Diagrama de componentes
- Diagrama de despliegue

Los diagramas más usados son los de casos de uso, clases y secuencia (además han sido usados en el desarrollo de este Trabajo de Fin de Carrera), por lo que se detallarán a continuación. Se utilizarán ejemplos de un sistema de venta de entradas de cine por Internet.

DIAGRAMA DE CASOS DE USO

Estos diagramas muestran de forma gráfica los casos de uso que tiene el sistema. Se entiende por caso de uso cada supuesta interacción con el sistema a desarrollar, donde se representan los requisitos funcionales. Resumiendo, se expresa lo que tiene que hacer un sistema y cómo debe hacerlo (representando la interacción con el exterior).

El sistema se representa como una caja rectangular con el nombre en su interior.

En un diagrama de casos de uso aparecen distintos elementos que se explicarán más tarde:

- Actores
- Casos de uso
- Relaciones

Los casos de uso se representan en el interior de dicha caja y los actores fuera. Las relaciones unen actores y casos de uso.

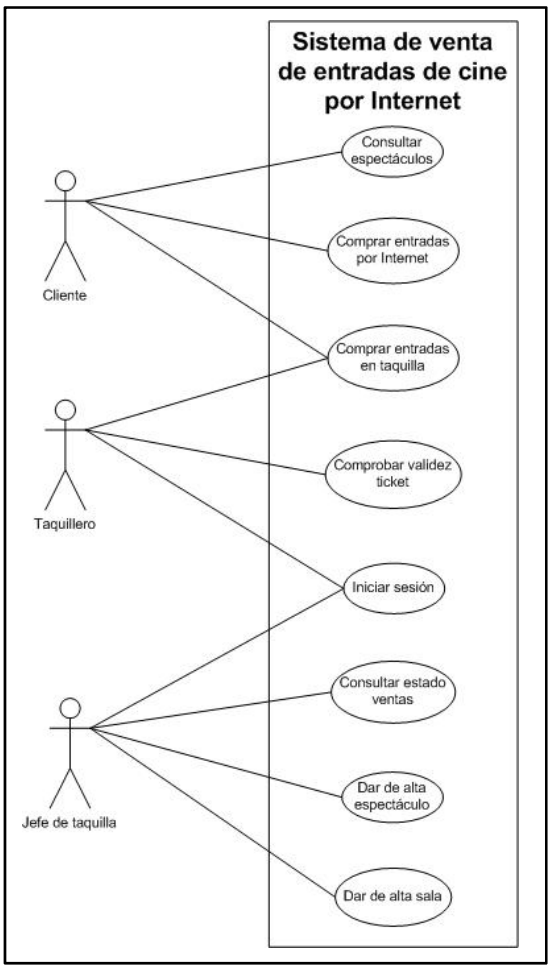

Fig 2-36 Ejemplo de diagrama de casos de uso

En la figura Fig 2-36 se muestra un ejemplo de casos de uso. Existen tres actores (clientes, taquilleros y jefes de taquilla) y distintas operaciones que pueden realizar.

Actores

Un actor es una entidad externa al sistema que actúa con él de alguna forma (usuarios, sensores, otros sistemas, etc.). Normalmente, los actores serán los usuarios (desde el punto de vista de la interacción un usuario es aquel que "usa" el sistema) del sistema y otros sistemas externos al que se está desarrollando. Un actor puede representar a muchos usuarios distintos y, de igual forma, un usuario puede actuar como distintos actores o, lo que es lo mismo, adoptar distintos roles. Un actor se puede

representar como un “monigote” o en una caja rectangular como se muestra en la figura Fig 2-37.

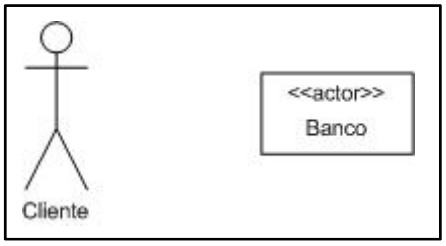

Fig 2-37 Distintas representaciones de actores

Casos de uso

Un caso de uso representa el comportamiento que tiene el sistema desde el punto de vista del usuario. Normalmente consiste en un conjunto de transacciones entre el sistema y los actores. Los casos de uso se pueden agrupar según funcionalidades semejantes o relacionadas, para facilitar su comprensión. Un caso de uso se representa mediante una elipse, con el nombre del caso de uso en su interior.

Relaciones

Las relaciones pueden darse:

- Entre actores y casos de uso: se trata de una relación de comunicación que indica que un actor interviene en el caso de uso. El actor aporta información para la realización del caso de uso y/o recibe información como resultado de la realización del mismo. Este tipo de relación puede ser unidireccional o bidireccional. En cualquier caso se representan con una línea continua entre las dos partes implicadas.
- Entre casos de uso: en este caso es una relación unidireccional. Los distintos tipos de relaciones existentes entre casos de uso son:
    - Incluye (<<*include*>>): se usa cuando se quiere expresar un comportamiento común a distintos casos de uso. Por ejemplo, (ver figura Fig 2-38) los casos de uso A y B tienen una parte común; ésta se puede representar como un tercer caso de uso y, como consecuencia, existirán dos relaciones “incluye”, de A a C y de B a C.

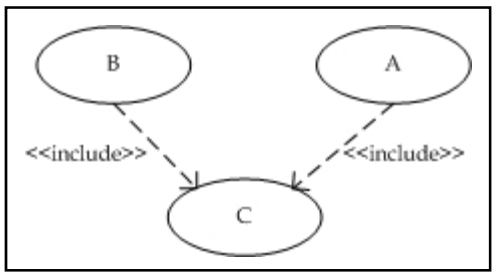

Fig 2-38 Ejemplo de una relación <<*include*>>

- Extiende (<<*extends*>>): se usa cuando se quiere expresar un comportamiento opcional en un caso de uso. Por ejemplo, existe el caso de uso A, el cual representa un comportamiento habitual en el sistema. Pero, dependiendo de algún factor, el caso de uso A puede presentar un comportamiento adicional representado en el caso de uso B. Se usaría una relación de tipo "extiende" para representar esta situación (ver figura Fig 2-39).

Fig 2-39 Ejemplo de una relación <<*extends*>>

Las relaciones entre casos de uso se representan con una flecha discontinua. Y con el tipo de relación como etiqueta. El sentido de la fechas en cada caso puede verse en los ejemplos representados en las figuras Fig 2-38 y Fig 2-39.

DIAGRAMA DE CLASES

Este tipo de diagrama es el más común para describir el diseño de los sistemas orientados a objetos. Un diagrama de clases representa un conjunto de clases, paquetes, interfaces y sus relaciones. Una clase describe un conjunto de objetos software que comparten propiedades similares (atributos) y un comportamiento común. Los objetos son instancias de las clases. En la figura Fig 2-40 se muestra un ejemplo de diagrama de clases como continuación al diagrama de casos de uso de la figura Fig 2-36.

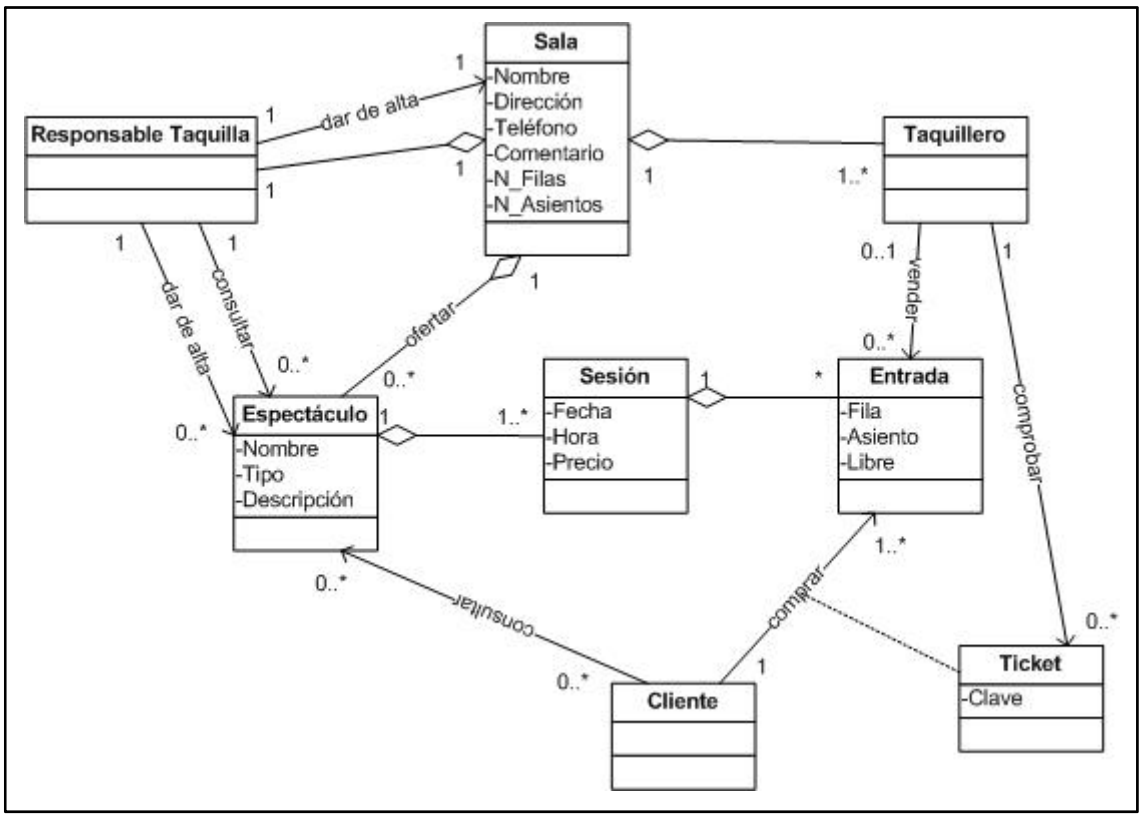

Fig 2-40 Ejemplo de diagrama de clases

Clases

La representación gráfica de una clase es un rectángulo, separado en tres zonas por líneas horizontales. En la zona superior se muestra el nombre de la clase (si la clase es abstracta se escribe en cursiva). La zona central contiene los atributos de la clase, uno por línea. La notación completa tiene el siguiente formato, aunque dependiendo del nivel de detalle se pueden obviar o no algunos campos:

visibilidad nombre : tipo = valor_inicial

La visibilidad se representará de la siguiente forma:

- '+': pública
- '-': privada
- '#': protegida

En la parte inferior del rectángulo se incluyen las operaciones propias de la clase, con el siguiente formato:

visibilidad nombre (lista_parámetros) : tipo_devuelto

La visibilidad se trata igual que en el caso de los atributos, y la lista de parámetros recoge los parámetros recibidos por la operación separados por comas y con el siguiente formato:

nombre : tipo_devuelto = valor_por_defecto

Paquetes

Los paquetes ofrecen un mecanismo general para la organización de los modelos, agrupando los elementos a modelar.

Cada paquete corresponde a un subconjunto del modelo y contiene clases, objetos, relaciones, componentes, diagramas asociados o incluso otros paquetes.

Un paquete se representa como una carpeta, con los elementos en su interior (figura Fig 2-41).

Fig 2-41 Representación de un paquete UML

Objetos

Un objeto tiene la misma representación que una clase, con la única diferencia que en la parte superior aparece además el nombre del objeto con la siguiente forma:

nombre_objeto : nombre_clase

Relaciones

Las relaciones se dan entre clases. Existen tres tipos fundamentales:

- Asociación: este tipo de relaciones simboliza un conjunto de enlaces entre objetos. La figura Fig 2-42 muestra el formato de representación de la relación de asociación. Además, para dar mayor información, se pueden mostrar elementos adicionales como:

- Nombre de la asociación: elemento opcional colocado junto a la línea.
- Rol: describe el papel del objeto o clase en la relación de asociación. Se coloca en los extremos asociados a las clases que desempeñan dicho rol.
- Multiplicidad: indica la cardinalidad de la relación. La letra 'n' y el símbolo '*' representan cualquier número. Se coloca en el mismo lugar que los roles.

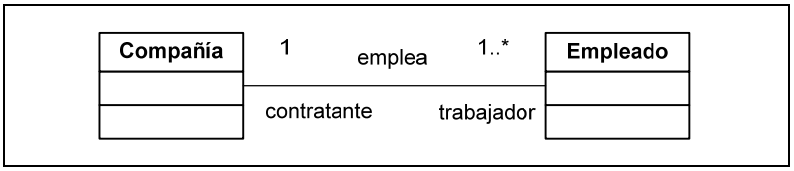

Fig 2-42 Ejemplo de relación de asociación

- Agregación: consiste en un tipo de relación jerárquica entre objetos, como se muestra en siguiente figura.

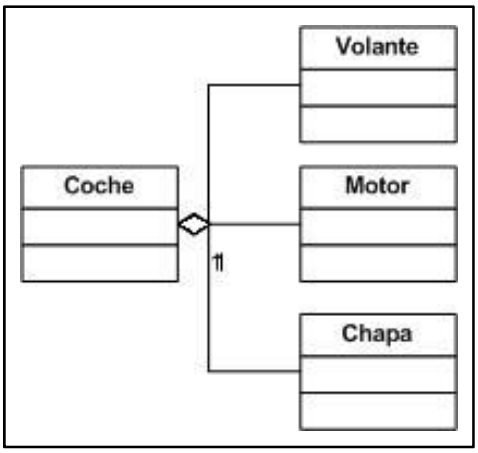

Fig 2-43 Ejemplo de relación de agregación

- Composición: es un tipo de relación de agregación en la que los objetos agregados no pueden existir si no existe el objeto que los agrega (ver la figura Fig 2-44).

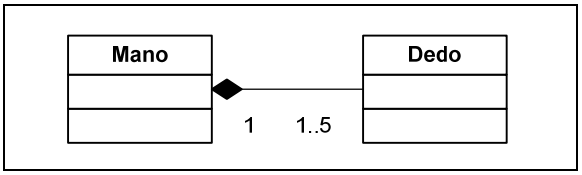

Fig 2-44 Ejemplo de relación de composición

- Dependencia: en este tipo de relación una clase necesita de otra para poder ofrecer determinados servicios, por lo que un cambio en la clase destino podría implicar cambios en la clase origen. La representación de esta relación se muestra en la figura Fig 2-45.

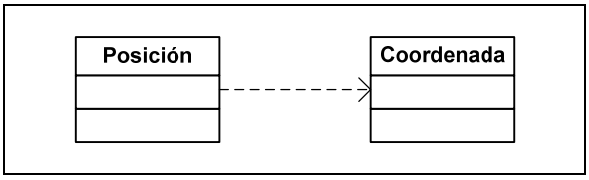

Fig 2-45 Ejemplo de relación de dependencia

Herencia

Es un elemento con el que se pueden incorporar atributos y operaciones de una clase a otra, añadiéndolos a los que ya tiene. Realmente es un tipo de relación entre clases, concretamente una relación “es-un” (ver figura Fig 2-46)

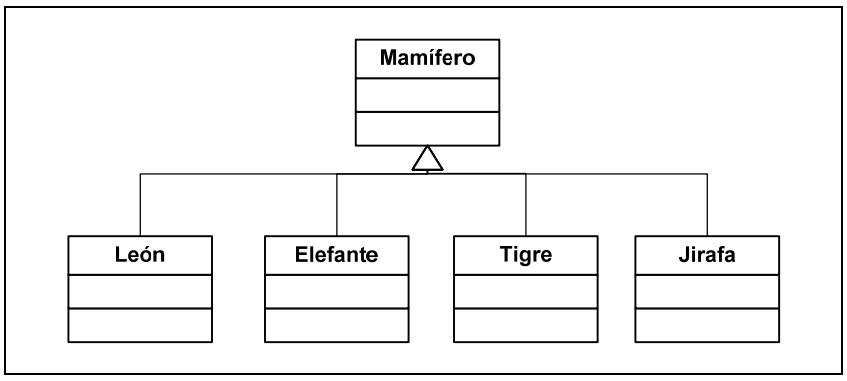

Fig 2-46 Ejemplo de herencia

DIAGRAMAS DE SECUENCIA

Este tipo de diagrama describe el comportamiento dinámico del sistema haciendo especial énfasis en la secuencia temporal de los mensajes entre objetos.

Un diagrama de secuencia es un diagrama en dos dimensiones. En la vertical se representa el tiempo (que avanza de arriba a abajo) y en el eje horizontal los diferentes objetos y los mensajes entre los mismos.

En la figura Fig 2-47 se muestra la representación de objetos y mensajes.

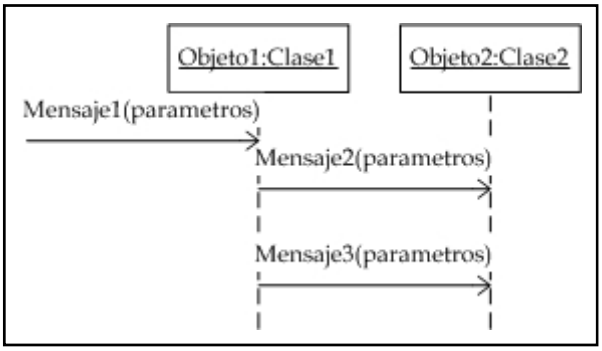

Fig 2-47 Representación de un diagrama de secuencia

Existen muchos más tipos de diagramas que los explicados en esta sección, cada uno de ellos está especializado en distintos aspectos del sistema a modelar. Por ejemplo, para modelar el comportamiento dinámico del sistema se utilizan los diagramas de colaboración, estados y actividades, mientras que para representar la implementación del sistema están los de componentes y despliegue.

2.2.10.2 Proceso de desarrollo

El lenguaje UML es fundamentalmente un conjunto de representaciones y diagramas y es bastante independiente del proceso de desarrollo que se siga en el software; sin embargo los mismos creadores de UML han propuesto su propia metodología de desarrollo: el Proceso Unificado de Desarrollo.

Este proceso se basa en componentes, es decir, el sistema a construir se basa en componentes software interconectados a través de interfaces bien definidos. Como es obvio, el Proceso Unificado utiliza el lenguaje UML para expresar gráficamente los esquemas del sistema.

El Proceso Unificado de Desarrollo se define porque es:

- iterativo e incremental: puesto que construir un sistema informático complejo es una tarea bastante larga y tediosa, este proceso propone la división del proyecto en varias fases. Actualmente se habla de ciclos de vida en los que se recorren todas las fases varias veces. Cada vez es una iteración, y cada iteración parte de la anterior, incrementando o revisando la funcionalidad implementada.

- dirigido por los casos de uso: se parte de un diseño basado en casos de uso a partir del cual se crean una serie de modelos de diseño que se implementan. Dichos modelos deben ser validados para estar conformes con los casos de uso. Los casos de uso también se utilizan para realizar pruebas sobre los componentes.
- centrado en la arquitectura: entendiendo por arquitectura software los aspectos estáticos y dinámicos más significativos del sistema.

Lo más importante es la característica de ser iterativo e incremental, lo que hace pensar que este proceso está pensado para el desarrollo de grandes proyectos.

3 ANÁLISIS DEL SISTEMA

3.1 ESPECIFICACIÓN DE REQUISITOS SOFTWARE

3.1.1 Introducción

Esta sección del trabajo de fin de carrera está dedicada a la Especificación de Requisitos Software (ERS) del sistema de mediación semántica, basado en ontologías, de Bases de Datos heterogéneas sobre tecnologías Grid. La estructura y formato de este apartado son los recomendados en el estándar "IEEE Recommended Practice for Software Requirements Specifications" (IEEE Estándar 830-1998) [59].

3.1.1.1 Propósito

El objetivo principal de esta sección es recoger una lista formal y exhaustiva de los requisitos software para la construcción del sistema. Dicha lista, además de definir todas las funcionalidades, características y restricciones que el sistema debe contener, servirá para verificar y validar la corrección e integridad del mismo.

Esta ERS va dirigida, por un lado a los desarrolladores que puedan necesitar conocer los requisitos que el sistema cumple en caso de querer modificarlo, ampliarlo u optimizarlo, y, por otro lado, al usuario final para que sepa lo que puede esperar del sistema.

3.1.1.2 Ámbito del sistema

El sistema desarrollado está pensado para ser usado en organizaciones pertenecientes al ámbito biomédico (centros de investigación, hospitales, etc), así como desde otras herramientas que necesiten de los servicios del sistema.

El sistema, fundamentalmente, permitirá realizar consultas a Bases de Datos heterogéneas y distribuidas, de forma remota y a través de Internet. Dichas Bases de

Datos deben ser “conocidas” previamente por el sistema través de un servicio de actualización de Bases de Datos.

El usuario o la aplicación tendrá a su disposición una serie de servicios Grid que estarán accesibles a través de la Web. Así, si el usuario o la aplicación desea hacer uso del servicio de consultas del sistema, tendrá la percepción de acceder a un único espacio de información independientemente de la estructura interna de cada una de las Bases de Datos, de su SGBD o de su localización. A su vez podrá conocer los resultados de su consulta:

- a través de una URL que apunte a una plataforma integrada donde se han colocado los resultados, o
- directamente, como respuesta al servicio llamado.

Este sistema puede ser considerado como un sistema útil para profesionales (en el campo de la medicina, ya que los datos con los que se trabaja son del dominio médico) que deseen realizar consultas a Bases de Datos heterogéneas. El usuario no necesitará conocer dónde se encuentran las fuentes de información ya que se podrá acceder a toda la información disponible en las Bases de Datos que se unan al sistema.

3.1.1.3 Acrónimos y Abreviaturas

Acrónimos

Acrónimo	Definiciones
ERS	Especificación de Requisitos Software
IEEE	Institute of Electrical and Electronics Engineers
SGBD	Sistema Gestor de Bases de Datos
CSV	Comma-separated values
DMS	Data Management System

Tabla 1 Acrónimos

Abreviaturas

Abreviatura	Definición
Std	Estándar

Tabla 2 Abreviaturas

3.1.1.4 Referencias

IEEE Recommended Practice for Software Requirements Specifications (IEEE Std. 830-1998).

3.1.1.5 Visión General

Este apartado, dedicado a la especificación de requisitos software se divide en tres subsecciones:

- La primera es la introducción, que proporciona una visión general de la ERS.
- La siguiente subsección contiene los factores que afectan al producto y a la ERS, sin especificar demasiados detalles. Se da una descripción general del sistema, realizando un resumen de las funciones que llevará a cabo el software, las restricciones, así como supuestos y dependencias que deben aplicarse al producto.
- En la tercera subsección se detallan las necesidades para el desarrollo. Se expone una lista detallada de los requisitos que debe satisfacer el sistema. También contiene los requisitos no funcionales del producto, como restricciones de rendimiento, de diseño o seguridad o interfaces externas.

3.1.2 Descripción Global

Este apartado contiene todos aquellos factores que afectan al producto y a la ERS. No se describen los requisitos en sí sino su contexto. A partir de este apartado se podrán comprender mejor los requisitos que se detallarán en la siguiente subsección, haciéndolos más sencillos de entender.

El sistema a desarrollar está compuesto por un conjunto de herramientas y servicios Grid para dar soporte a la integración de Bases de Datos heterogéneas, solventando heterogeneidades a nivel de instancia y de esquema.

3.1.2.1 Perspectiva del Producto

Este sistema se encuentra dentro del conjunto de herramientas que conforman la plataforma de trabajo de ACGT (*Advancing Clinico Genomic Trial son Cancer*), por lo que, en cierto sentido, es dependiente del funcionamiento de los servicios de acceso a datos y de la herramienta de *mapping*, los cuales acceden a las fuentes de información y suministran las correspondencias existentes entre términos conceptuales y físicos, respectivamente.

3.1.2.2 Funciones del Producto

Las principales funciones que realiza el software creado son:

- Proporcionar al usuario la posibilidad de realizar consultas a Bases de Datos heterogéneas y distribuidas.
- Ofrecer al usuario la posibilidad de obtener los resultados en forma de fichero CSV o a través de una URL que apunte a una plataforma integrada donde se han colocado los resultados.
- Permitir la consulta y la actualización de las Bases de Datos disponibles.
- Proporcionar la posibilidad de informarse sobre los datos que pueden ser consultados.
- Creación de esquemas de Bases de Datos en SQL.

3.1.2.3 Características del Usuario

El sistema tiene dos tipos de usuarios. Por un lado se encuentra el administrador, y por otro, los usuarios finales:

- El administrador del sistema es el usuario encargado de instalar el sistema. También se encargará de registrar nuevas Bases de Datos en el sistema así como de realizar cualquier modificación sobre las ya existentes. El administrador deberá tener conocimientos técnicos así como conocimientos sobre SGBD ya que

se proporcionará la URL de conexión a cada uno de los *wrappers* (envoltorio) de las Bases de Datos.

- Los usuarios finales son profesionales que desean obtener resultados de consultas a Bases de Datos heterogéneas y distribuidas. El usuario final usará el sistema a través de la Web, llamando a los distintos servicios que estarán disponibles, o bien a través de una sencilla interfaz (por ejemplo usando el ratón y menús), o bien accediendo a ellos directamente. Por esta razón el usuario final deberá tener conocimientos de informática, tanto de su sistema operativo, como de navegación Web, para el manejo de la interfaz; serán especialmente útiles, conocimientos sobre Servicios Web si se accede directamente al servicio. Puesto que el sistema se desarrolla en el ámbito de de la investigación biomédica, será necesario tener conocimientos propios de medicina y biología. Los datos incluidos en las Bases de Datos están relacionados con esos temas, pues los usuarios finales serán principalmente médicos y biólogos.

3.1.2.4 Restricciones

Debido al uso del lenguaje de programación Java, el sistema desarrollado es independiente tanto de la plataforma de ejecución como del sistema operativo en el que se ejecute. Se necesitará por tanto el intérprete adecuado para cada plataforma y sistema operativo para la ejecución de los *bytecodes* generados por el compilador Java.

Serán necesarias todas las clases Java incluidas, al menos en la versión del JSDK 1.6.3.

Además será necesaria la instalación tanto de Globus Toolkit 4.0.2. y 4.0.3. en su versión para servicios Web, como de OGSA-DAI (en la versión compatible con las nombradas de Globus Toolkit y también en su versión para servicios Web).

3.1.2.5 Suposiciones y dependencias

Se supone que todos los datos proporcionados a los servicios son correctos, en concreto los *mappings* que se actualicen en el mismo; lo cual incluye las direcciones donde se encuentran los *wrappers* a los que se refieren y los *paths* conceptuales y físicos contenidos en ellos.

3.1.3 Requisitos Específicos

En esta sección se detallan los requisitos funcionales que deberá recoger el funcionamiento del sistema.

3.1.3.1 Requisitos de Interfaces Externos

En este apartado se recogen los interfaces que el sistema debe llevar a cabo para interactuar con otros sistemas

Interfaz de usuario

No se han definido

Interfaces Hardware

[REQ#01] El sistema deberá funcionar en distintas plataformas hardware.

Interfaces Software

[REQ#02] El sistema deberá funcionar correctamente en diversas plataformas software (Windows, Linux, etc.)

[REQ#03] El sistema debe comunicarse con *wrappers* que estén en otras máquinas.

[REQ#04] El sistema estará basado en Globus Toolkit y OGSA-DAI para la construcción de los Servicios Grid.

Interfaces de Comunicación

[REQ#05] El sistema debe estar basado en los Servicios Grid y, por lo tanto, el estándar XML y el protocolo SOAP se utilizarán para representar e intercambiar información.

3.1.3.2 Requisitos Funcionales

En esta sección se detallan las acciones (funciones, servicios, etc) que deberá llevar a cabo el sistema.

Para exponer los requisitos funcionales se ha seguido el orden de los objetivos, entendiéndose como tal aquellos servicios que se desea que ofrezca el sistema.

Consultas a Bases de Datos

[REQ#06] El sistema reconocerá el lenguaje SPARQL para las consultas a las Bases de Datos.

[REQ#07] El sistema será capaz de reconocer a qué Bases de Datos van dirigidas las consultas que reciba.

[REQ#08] El sistema dividirá la consulta en subconsultas dependiendo de la parte que respecte a cada Base de Datos.

[REQ#09] El sistema traducirá las consultas desde términos conceptuales a físicos.

[REQ#10] El sistema se comunicará con los respectivos *wrappers* para lanzar la subconsulta generada.

[REQ#11] El sistema esperará por los resultados de cada *wrapper*.

[REQ#12] El sistema integrará los resultados devueltos por los *wrappers* resolviendo las heterogeneidades semánticas que surgieran.

[REQ#13] El sistema extraerá de los archivos de *mapping* toda la información respectiva a traducciones de *paths*.

[REQ#14] El sistema devolverá los resultados en formato .csv.

[REQ#15] El sistema devolverá los resultados a la consulta en mismos términos en que fue hecha, con los nombres de los atributos originales.

Actualización de resultados en un repositorio integrado

[REQ#16] El sistema permitirá la opción de actualizar los resultados a una consulta en un repositorio integrado (DMS), subiendo los mismos en un fichero .csv.

[REQ#17] El sistema contará con la acreditación necesaria (certificados) y la usará correctamente, para poder acceder a un repositorio seguro perteneciente a la plataforma.

[REQ#18] En caso de actualizarse los resultados en el repositorio integrado, el sistema generará metadatos sobre los resultados.

[REQ#19] Los metadatos generados deberán actualizarse en el repositorio integrado, en un único archivo, proporcionando la información necesaria para relacionar los resultados con sus correspondientes metadatos.

[REQ#20] El sistema devolverá a la petición de actualización de actualización de resultados en el repositorio integrado: el identificador de la carpeta en la que se han colocado los resultados y los metadatos, y el identificador del fichero que contiene los resultados.

Actualización de *mappings* en el sistema

[REQ#21] El sistema será capaz de recibir ficheros de *mapping* desde el exterior, proporcionando un único servicio para la actualización de un nuevo fichero o para la modificación de uno ya existente.

[REQ#22] Será el propio sistema el que distinga si se trata de la creación de un *mapping* nuevo o de la modificación de uno ya existente.

[REQ#23] El sistema creará un nuevo fichero en caso de que el *mapping* sea nuevo e informará a las aplicaciones de mediación semántica de dicha creación.

[REQ#24] El sistema modificará el fichero de *mapping* que ya exista en caso de tratarse de una modificación.

Listado de *wrappers*

[REQ#25] El sistema proporcionará los nombres y direcciones de los *wrappers* que es capaz de consultar.

Listado de *mappings*

[REQ#26] El sistema proporcionará los *mappings* que es capaz de consultar.

Creación del esquema de una Base de Datos

[REQ#27] El sistema será capaz de crear esquemas .n3 a partir de Bases de Datos Relacionales SQL.

3.1.3.3 Requisitos de Rendimiento

En esta sección se describen los requisitos numéricos o medibles acerca del sistema o de la interacción del usuario con el software.

[REQ#28] Se ofrecerán distintos servicios Grid al usuario.

[REQ#29] El software debe poder ser accedido simultáneamente, proporcionando el mismo servicio a cada usuario.

[REQ#30] El tiempo de respuesta a una consulta a Bases de Datos heterogéneas no debe ser "elevado", así que, por lo que respecta a la aplicación de mediación semántica, cuando ésta deba actualizar los resultados en el repositorio integrado, no deberá esperar por la respuesta del mismo.

3.1.3.4 Requisitos de Diseño

En estos requisitos se contemplan restricciones de diseño impuestas por estándares, plataformas hardware y software, etc.

[REQ#31] El sistema hará uso de la tecnología Globus Toolkit para proporcionar Servicios Grid al exterior.

[REQ#32] El sistema hará uso de la tecnología proporcionada por la plataforma OGSA-DAI.

[REQ#33] El sistema guardará un "log" de las acciones realizadas contra el sistema.

3.2 CASOS DE USO DEL SISTEMA

Esta sección no se encuentra recogida dentro del estándar IEEE Std. 830-1998. Sin embargo, se ha incluido en este apartado dedicado a la especificación de requisitos

software por su relación con el mismo. Los casos de uso se han extraído directamente de los requisitos mencionados en la sección anterior.

Los casos de uso son una forma de expresar cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo determinado. Dicho de otra forma, expresan los requisitos funcionales del sistema desde la perspectiva del usuario.

A continuación se describen los actores que interactúan con el sistema, los objetivos de cada uno de ellos (representados en un diagrama de casos de uso) y una exposición detallada de cada uno de los casos de uso.

3.2.1 Actores

En las secciones anteriores se ha definido que existen dos tipos de usuarios o actores principales en el sistema: el administrador y el usuario final. También existen actores de apoyo, los cuales proporcionan algún servicio al sistema. A continuación se describe cada uno de ellos:

3.2.1.1 Actores Principales

Administrador: es el encargado de instalar el sistema, configurarlo, hacer operativa una nueva Base de Datos en el sistema, crear un esquema de la misma o modificar alguna de las Bases de Datos del sistema.

Usuario final: usa el sistema a través de la Web, de dos formas posibles (como se comentó en la sección 4.1.2.3): o bien llamando directamente a los servicios que estarán disponibles, o bien a través de una sencilla interfaz.

Los perfiles de los dos actores principales fueron descritos en la sección 4.1.2.3.

MappingTool: se trata de otro sistema que utiliza el propio para listar los *wrappers* que conoce el propio sistema así como para llamar al servicio de actualización de *mappings*.

QueryTool: se trata de otro sistema que utiliza el propio para realizar consultas y recoger resultados así como para utilizar el servicio de listado de los *mappings* disponibles en el propio sistema.

3.2.1.2 Actores de Apoyo

***Wrapper*:** los *wrappers* son aquellos sistemas que "envuelven" una única Base de Datos. De esta forma, recibirán una consulta en SPARQL sobre la Base de Datos (sea cual sea el lenguaje bajo el que estén implementadas) y devolverán los correspondientes resultados a la misma.

DMS: el *Data Management System* es una aplicación middleware de gestión de datos orientada al Grid. Su papel desde el punto del sistema consiste en recibir los archivos que éste le envíe y devolver el identificador con el que han sido almacenados.

3.2.2 Diagrama de Casos de Uso

Un diagrama de casos de uso muestra los límites del sistema, lo que permanece fuera de él y cómo se utiliza. Sirve como herramienta de comunicación que resume el comportamiento de un sistema y sus actores.

La figura Fig 3-1 muestra el diagrama de casos de uso del sistema.

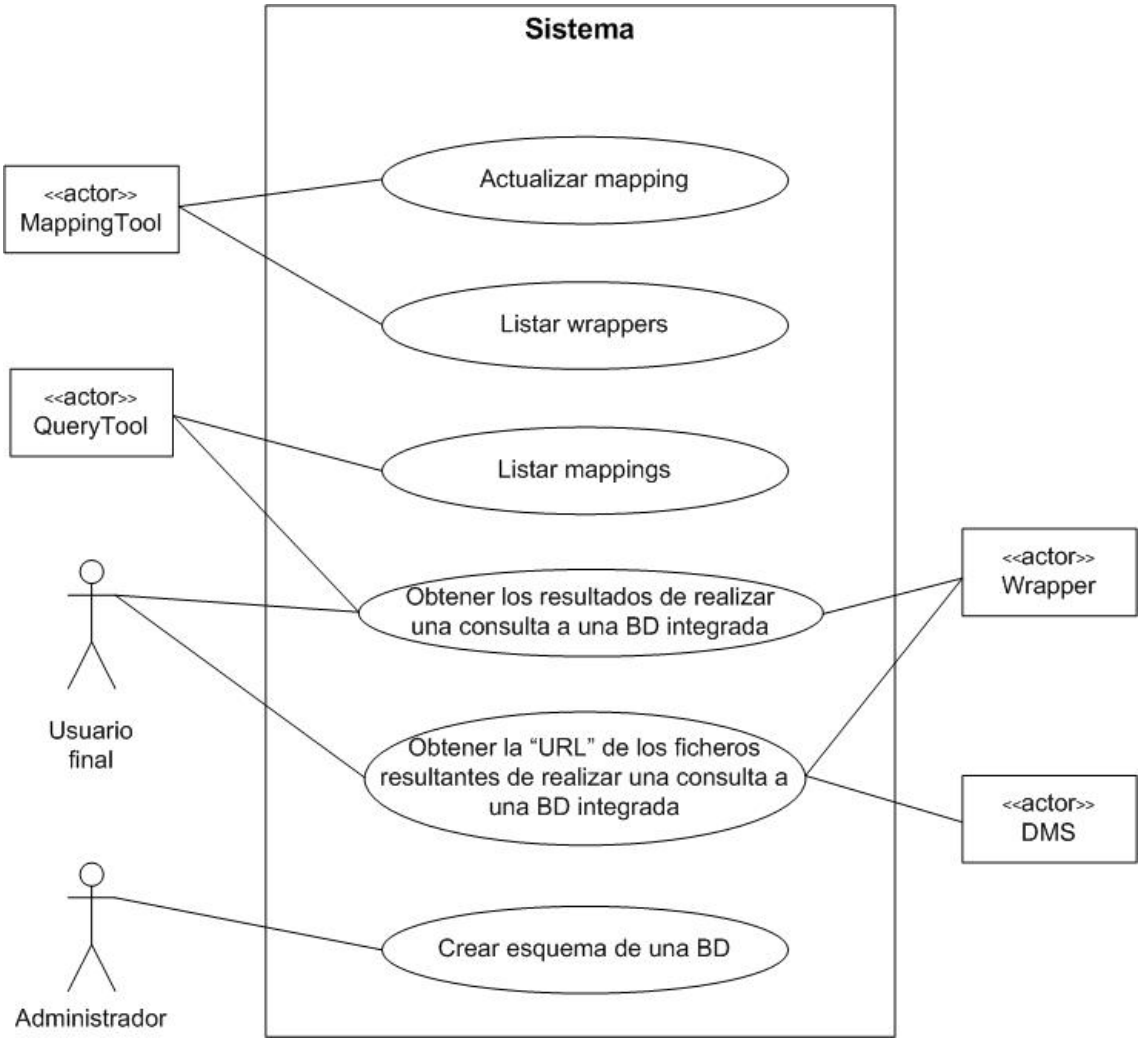

Fig 3-1 Diagrama de Casos de Uso

3.2.3 Casos de Uso

A continuación se explicarán de forma detallada cada uno de los casos de uso identificados, en formato completo.

3.2.3.1 Caso de Uso "Actualizar mapping"

Objetivo: Actualizar la lista de *mappings* disponibles en el sistema.

Nivel: Primario.

Tipo: Esencial.

Actor principal: *MappingTool*

Actores secundarios: -

Precondiciones: Ninguna

Referencias: [REQ#01], [REQ#02], [REQ#04], [REQ#05], [REQ#21], [REQ#22], [REQ#23], [REQ#24], [REQ#28], [REQ#29], [REQ#31], [REQ#32].

Escenario principal:

1. La *MappingTool* envía un *mapping* para ser actualizado/creado en el sistema.
2. El sistema actualiza/crea el *mapping*.

Escenarios alternativos:

*. En cualquier momento el sistema falla.

1. El sistema se reinicia.

3.2.3.2 Caso de Uso “Listar wrappers”

Objetivo: Obtener una lista con los servicios *wrapper* disponibles en el sistema.

Nivel: Primario.

Tipo: Esencial.

Actor principal: *MappingTool*

Actores secundarios: -

Precondiciones: Ninguna.

Referencias: [REQ#01], [REQ#02], [REQ#04], [REQ#05], [REQ#25], [REQ#28], [REQ#29], [REQ#31], [REQ#32].

Escenario principal:

1. La *MappingTool* solicita un listado de los *wrappers* disponibles en el sistema.
2. El sistema envía a la *MappingTool* el listado.
3. La *MappingTool* recibe el listado.

Escenarios alternativos:

*. En cualquier momento el sistema falla.

1. El sistema envía un mensaje de error a la *MappingTool*.
2. El sistema se reinicia.

3.2.3.3 Caso de Uso “Listar mappings”

Objetivo: Obtener una lista detallada de todos los *mappings* del sistema.

Nivel: Primario.

Tipo: Esencial.

Actor principal: *QueryTool*

Actores secundarios: -

Precondiciones: Ninguna.

Referencias: [REQ#01], [REQ#02], [REQ#04], [REQ#05], [REQ#26], [REQ#28], [REQ#29], [REQ#31], [REQ#32].

1. La *QueryTool* solicita una lista de los *mappings* disponibles en el sistema.
2. El sistema envía a la *QueryTool* el listado.
3. La *QueryTool* recibe el listado.

Escenarios alternativos:

*. En cualquier momento el sistema falla.

1. El sistema envía un mensaje de error a la *QueryTool*.
2. El sistema se reinicia.

3.2.3.4 Caso de Uso “Obtener los resultados de realizar una consulta a una BD integrada”

Objetivo: Realizar una consulta sobre un repositorio y obtener los resultados correspondientes.

Nivel: Primario.

Tipo: Esencial.

Actor principal: *QueryTool*, usuario final.

Actores secundarios: *Wrapper*.

Precondiciones:

Referencias: [REQ#01], [REQ#02], [REQ#03], [REQ#04], [REQ#05], [REQ#06], [REQ#07], [REQ#08], [REQ#09], [REQ#10], [REQ#11], [REQ#12], [REQ#13], [REQ#14], [REQ#28], [REQ#29], [REQ#31], [REQ#32], [REQ#33].

Escenario principal:

1. El usuario final o la *QueryTool* realiza una consulta al sistema.
2. El sistema envía los resultados al usuario final o a la *QueryTool*.
3. El usuario final o la *QueryTool* recibe los resultados.

Escenarios alternativos:

*. En cualquier momento el sistema falla.

1. El sistema envía un mensaje de error.
2. El sistema se reinicia.

3.2.3.5 Caso de Uso “Obtener la URL de los ficheros resultantes de realizar una consulta a una BD integrada”

Objetivo: Realizar una consulta sobre un repositorio y obtener un identificador referente al lugar donde se han almacenado los resultados.

Nivel: Primario.

Tipo: Esencial.

Actor principal: Usuario final.

Actores secundarios: *Wrapper*, DMS.

Precondiciones:

Referencias: [REQ#01], [REQ#02], [REQ#03], [REQ#04], [REQ#05], [REQ#06], [REQ#07], [REQ#08], [REQ#09], [REQ#10], [REQ#11], [REQ#12], [REQ#13], [REQ#14], [REQ#15], [REQ#16], [REQ#17], [REQ#18], [REQ#19], [REQ#20], [REQ#28], [REQ#29], [REQ#30], [REQ#31], [REQ#32], [REQ#33].

Escenario principal:

1. El usuario final realiza una consulta al sistema y proporciona un nombre de fichero para almacenar los resultados en el DMS.
2. El sistema obtiene los resultados de la consulta.
3. El sistema envía los resultados al repositorio DMS.
4. El DMS envía al sistema la identificación de los ficheros que ha almacenado.
5. El sistema envía los identificadores al usuario final.
6. El usuario final recibe los resultados.

Escenarios alternativos:

*. En cualquier momento el sistema falla.

1. El sistema envía un mensaje de error al usuario.
2. El sistema se reinicia.

4a. El DMS envía un mensaje de error al sistema porque el nombre de fichero ya existe.

1. El sistema envía un mensaje de error al usuario.
2. El sistema se reinicia.

3.2.3.6 Caso de Uso “Crear esquema de una BD”

Objetivo: Creación de un esquema .n3 de una Base de Datos.

Nivel: Primario.

Tipo: Esencial.

Actor principal: Administrador.

Actores secundarios: -

Precondiciones: Ninguna.

Referencias: [REQ#01], [REQ#02], [REQ#27].

Escenario principal:

1. El administrador solicita la creación del esquema de una Base de Datos.

2. El sistema crea un archivo con el esquema.

Escenarios alternativos:

*. En cualquier momento el sistema falla.

1. El caso de uso finaliza.

2a. La Base de Datos no existe o alguno de los parámetros proporcionados es erróneo.

1. El caso de uso finaliza.

3.2.4 Diagramas de secuencia del sistema

En este apartado se muestra el comportamiento del sistema, tratando a éste como una caja negra, es decir, se oculta su funcionamiento interno, para describir únicamente su comportamiento visto de fuera.

3.2.4.1 Diagrama de Secuencia del Caso de Uso "Actualizar mapping"

Fig 3-2 Diagrama de secuencia del caso de uso "Actualizar mapping"

3.2.4.2 Diagrama de Secuencia del Caso de Uso "Listar wrappers"

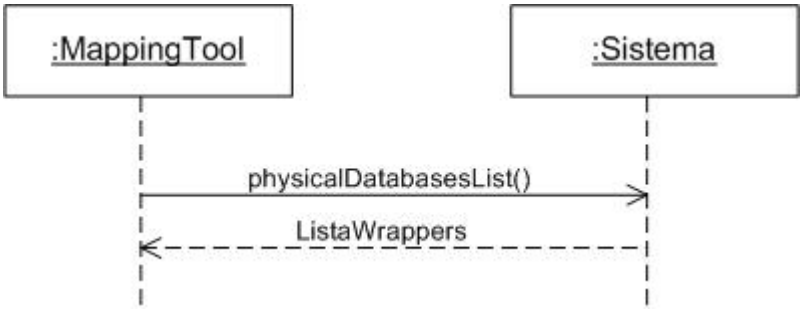

Fig 3-3 Diagrama de secuencia del caso de uso "Listar wrappers"

3.2.4.3 Diagrama de Secuencia del Caso de Uso "Listar mappings"

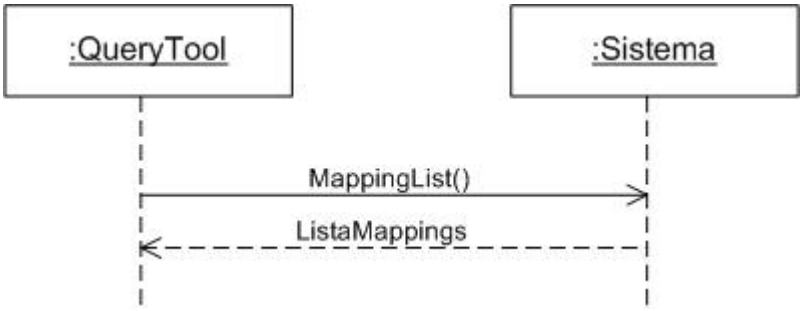

Fig 3-4 Diagrama de secuencia del caso de uso "Listar mappings"

3.2.4.4 Diagrama de Secuencia del Caso de Uso "Obtener los resultados de realizar una consulta a una BD integrada"

El actor principal de este caso de uso puede ser el usuario final o la *QueryTool*.

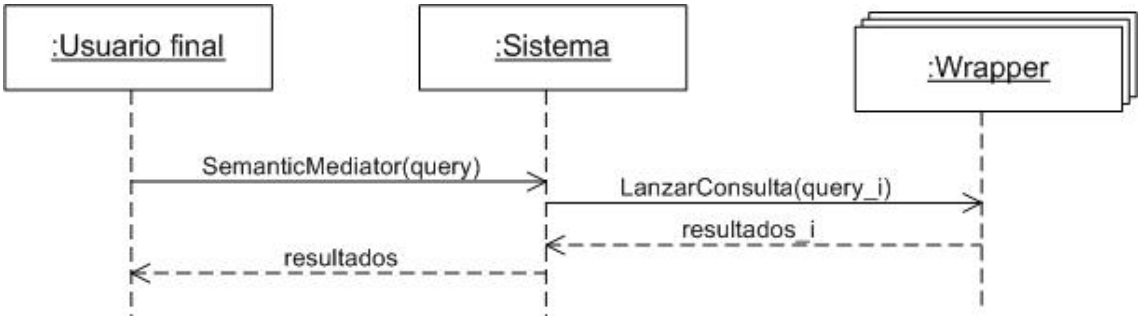

Fig 3-5 Diagrama de secuencia del caso de uso "Obtener los resultados de realizar una consulta a una BD integrada"

3.2.4.5 Diagrama de Secuencia del Caso de Uso "Obtener la URL de los ficheros resultantes de realizar una consulta a una BD integrada"

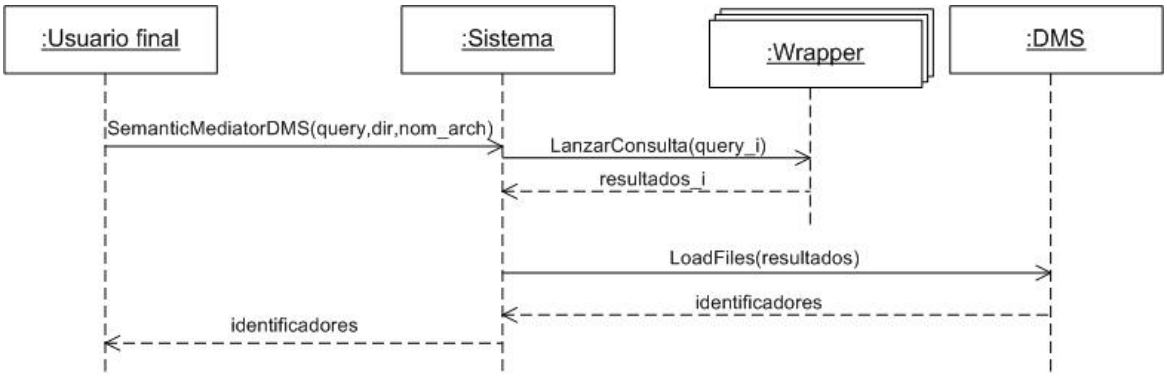

Fig 3-6 Diagrama de secuencia del caso de uso "Obtener la URL de los ficheros resultantes de realizar una consulta a una BD integrada"

3.2.4.6 Diagrama de Secuencia del Caso de Uso "Crear esquema de una BD"

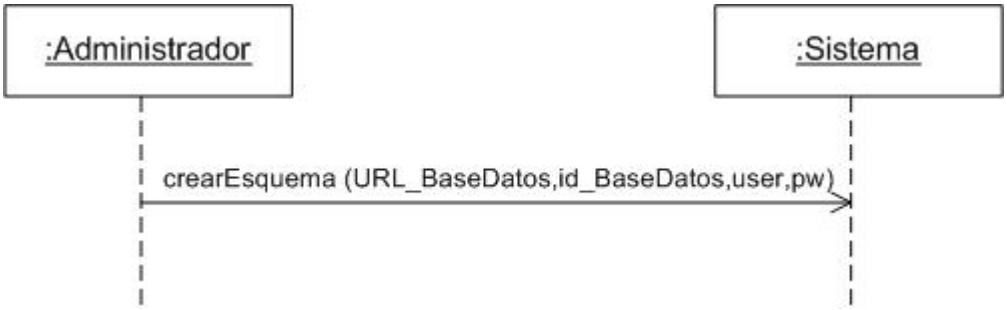

Fig 3-7 Diagrama de secuencia del caso de uso "Crear esquema de una BD"

3.2.5 Contratos de las Operaciones del Sistema

A continuación se describe el comportamiento que se espera de cada una de las operaciones del sistema que se han visto en los diagramas de secuencia de los casos de uso. Las siguientes tablas, muestran los contratos de las operaciones.

Nombre:	**Contrato CO1: *updateMappingList***
Operación:	*updateMappingList (mapping)*
Responsabilidades:	El sistema actualiza la lista de mappings que utiliza con el mapping “mapping”.
Precondiciones:	Ninguna.
Postcondiciones:	El sistema queda preparado para que se realicen consultas sobre el nuevo mapping.
Referencias cruzadas:	Caso de uso *Actualizar mapping.*
Salidas:	Ninguna.
Excepciones:	Ninguna.

Tabla 3 Contrato de la operación “updateMappingList”

Nombre:	**Contrato CO2: *physicalDatabasesList***
Operación:	*physicalDatabasesList ()*
Responsabilidades:	El sistema lista los wrappers que conoce.
Precondiciones:	Ninguna.
Postcondiciones:	Ninguna.
Referencias cruzadas:	Caso de uso *Listar wrappers.*
Salidas:	El sistema muestra la lista de los wrappers sobre los que se pueden realizar consultas.
Excepciones:	Ninguna.

Tabla 4 Contrato de la operación “physicalDatabasesList”

Nombre:	**Contrato CO3: *MappingList***
Operación:	*MappingList ()*
Responsabilidades:	El sistema lista los mappings que conoce.
Precondiciones:	Ninguna.
Postcondiciones:	Ninguna.
Referencias cruzadas:	Caso de uso *Listar mappings.*
Salidas:	El sistema muestra la lista de los mappings sobre los que se pueden realizar consultas.
Excepciones:	Ninguna.

Tabla 5 Contrato de la operación "MappingList"

Nombre:	**Contrato CO4: *SemanticMediator***
Operación:	*SemanticMediator (query)*
Responsabilidades:	El sistema procesa una consulta y devuelve los resultados de la misma.
Precondiciones:	Ninguna.
Postcondiciones:	Ninguna.
Referencias cruzadas:	Caso de uso *Obtener los resultados de realizar una consulta a una BD integrada.*
Salidas:	El sistema devuelve los resultados a la consulta realizada.
Excepciones:	Ninguna.

Tabla 6 Contrato de la operación "SemanticMediator"

Nombre:	**Contrato CO5: *SemanticMediatorDMS***
Operación:	*SemanticMediatorDMS (query, dir, nom_arch)*
Responsabilidades:	El sistema procesa una consulta, almacena los resultados en un repostorio, en el directorio y con el nombre de archivo especificado y devuelve los identificadores del directorio y el archivo en el repositorio DMS.
Precondiciones:	Ninguna.
Postcondiciones:	Ninguna.
Referencias cruzadas:	Caso de uso *Obtener la URL de los ficheros resultantes de realizar una consulta a una BD integrada.*
Salidas:	El sistema devuelve los identificadores del directorio y archivo en el repositorio DMS.
Excepciones:	Ninguna.

Tabla 7 Contrato de la operación "SemanticMediatorDMS"

Nombre:	**Contrato CO6: *crearEsquema***
Operación:	*crearEsquema (URL_BaseDatos, id_BaseDatos, user, password)*
Responsabilidades:	El sistema crea un esquema "n3" de una base de datos.
Precondiciones:	Ninguna.
Postcondiciones:	Ninguna.
Referencias cruzadas:	Caso de uso *Crear esquema de una BD.*
Salidas:	El sistema devuelve el esquema "n3" de la base de datos que se ha proporcionado.
Excepciones:	Ninguna.

Tabla 8 Contrato de la operación "crearEsquema"

3.3 MODELO DE DOMINIO

El modelo conceptual de dominio derivado de los casos de uso anteriores se muestra en la figura Fig 3-8:

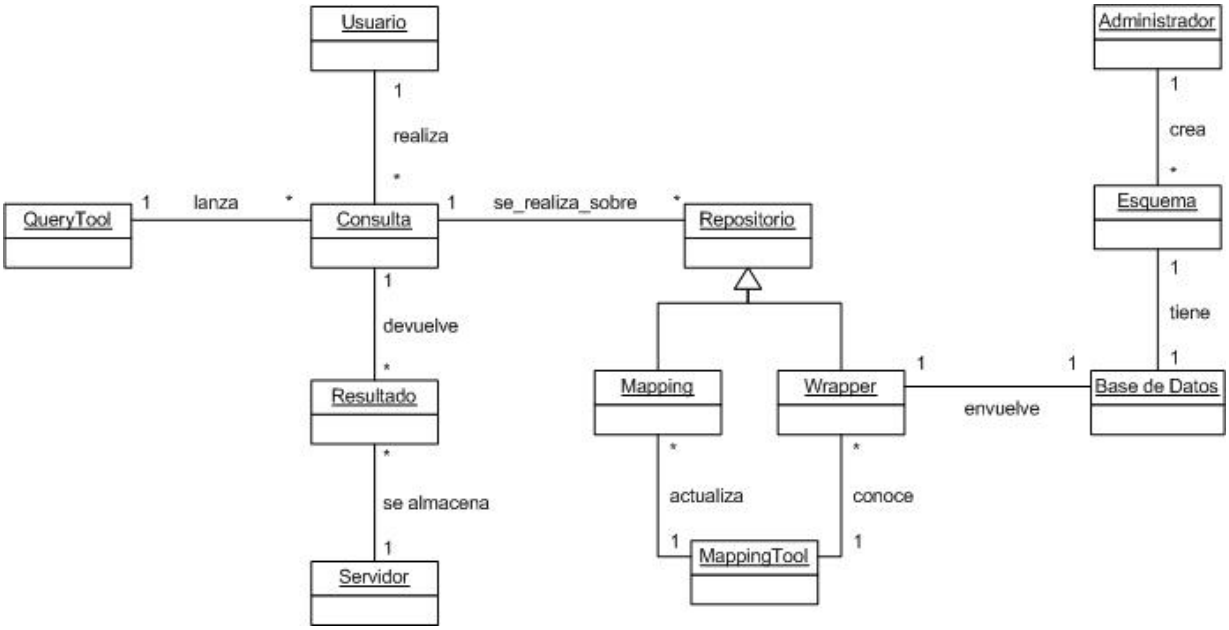

Fig 3-8 Modelo conceptual de dominio

Por motivos de claridad, la lista de atributos de los conceptos se muestra a continuación:

- **QueryTool**: id_consulta.
- **Consulta**: variables_Interes, restricciones, filtros.
- **Resultado**: datos, metadatos.
- **Servidor**: URL, certificados.
- **Repositorio**: URL, id_Wrapper.
- **Mapping**: entries.
- **Wrapper**: URL
- **MappingTool**: mapping.
- **Esquema**: baseDatos
- **Base de Datos**: localización

4 DISEÑO E IMPLEMENTACIÓN DEL SISTEMA

4.1 INTRODUCCIÓN

En este capítulo se describe de forma detallada el diseño y la implementación de la solución propuesta; para ello se mostrarán los diagramas de clases software y los diagramas de interacción.

A pesar de que todo el sistema trate de resolver la mediación semántica entre usuarios y bases de datos, la implementación del mismo se encuentra dividida en dos subsistemas o dos módulos. Por una parte, el encargado de tratar lo relacionado con los archivos de *mapping* (una serie de servicios para actualizar, listar o detallar los archivos de *mapping*) y por otra el módulo encargado de realizar o de gestionar la mediación semántica.

4.2 DIAGRAMAS DE INTERACCIÓN

En el lenguaje de modelado UML se utilizan los diagramas de interacción para mostrar el modo en el que interaccionan los objetos por medio de mensajes entre ellos. Hay dos tipos de diagramas de interacción: diagramas de colaboración y de secuencia. En esta sección se muestran los diagramas de secuencia, los cuales se han elegido porque facilitan la visualización de la secuencia de mensajes.

El diagrama de secuencia de la figura Fig 4-1 muestra como el sistema procesa una consulta para posteriormente subir los resultados y los metadatos a un repositorio integrado (DMS). Por cuestiones de espacio y claridad se han omitido parámetros de algunas operaciones.

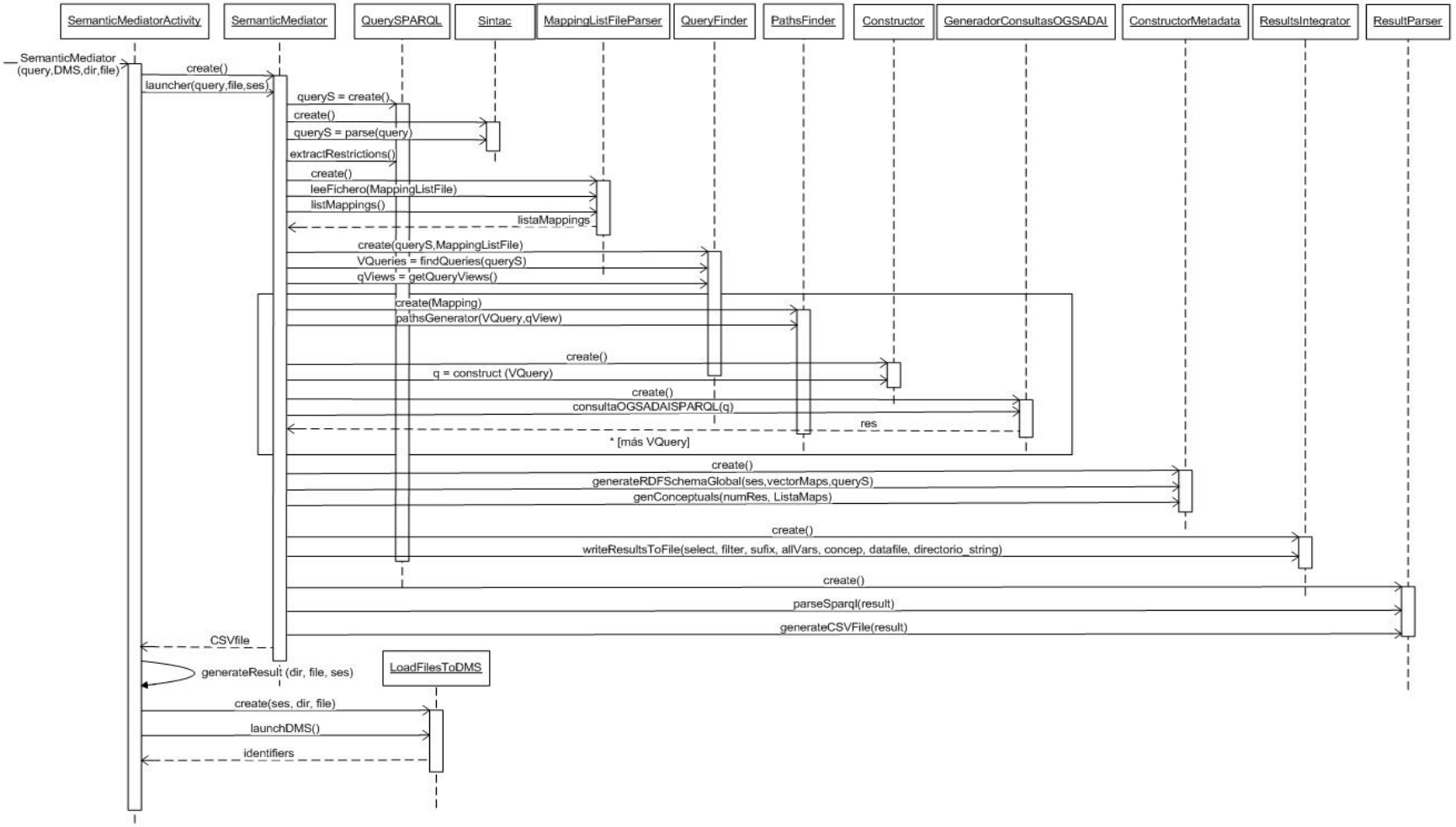

Fig 4-1 Diagrama de secuencia de la operación que realiza una consulta y actualiza los resultados en el DMS

4.3 DIAGRAMA DE PAQUETES

En esta sección se incluye el diagrama de paquetes del sistema (Fig 4-2). Un diagrama de paquetes muestra los paquetes existentes y sus relaciones, entendiendo por paquete una agrupación de elementos relacionados. Este tipo de diagrama es especialmente útil si el modelo a representar es muy grande, como es el caso.

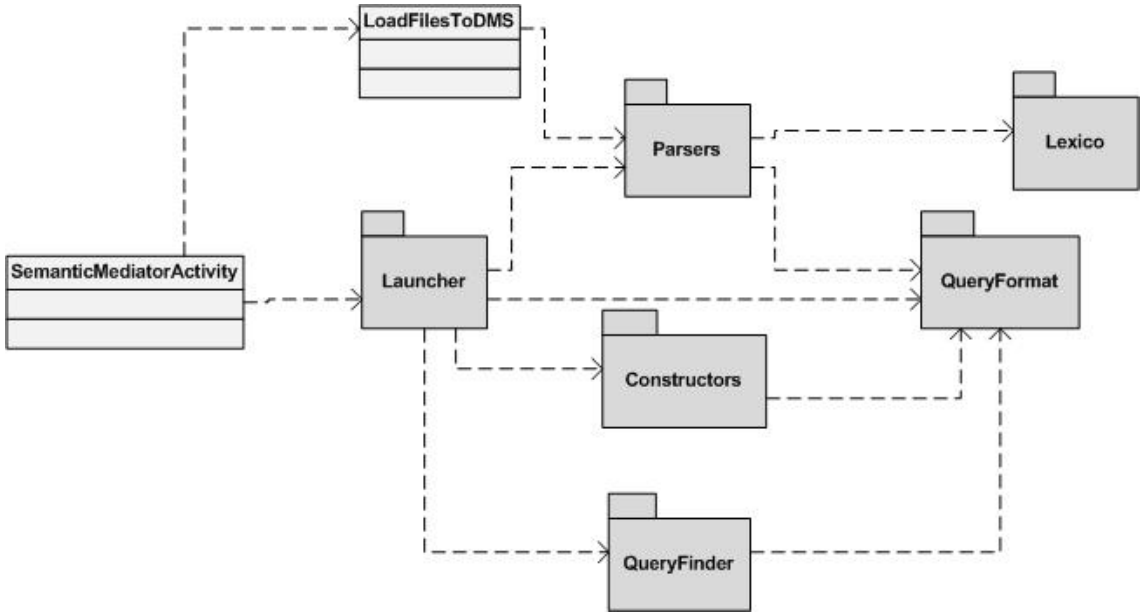

Fig 4-2 Diagrama de paquetes

En la siguiente sección se especifica, en el diagrama de clases del sistema, el contenido de cada uno de los paquetes.

4.4 DIAGRAMA DE CLASES

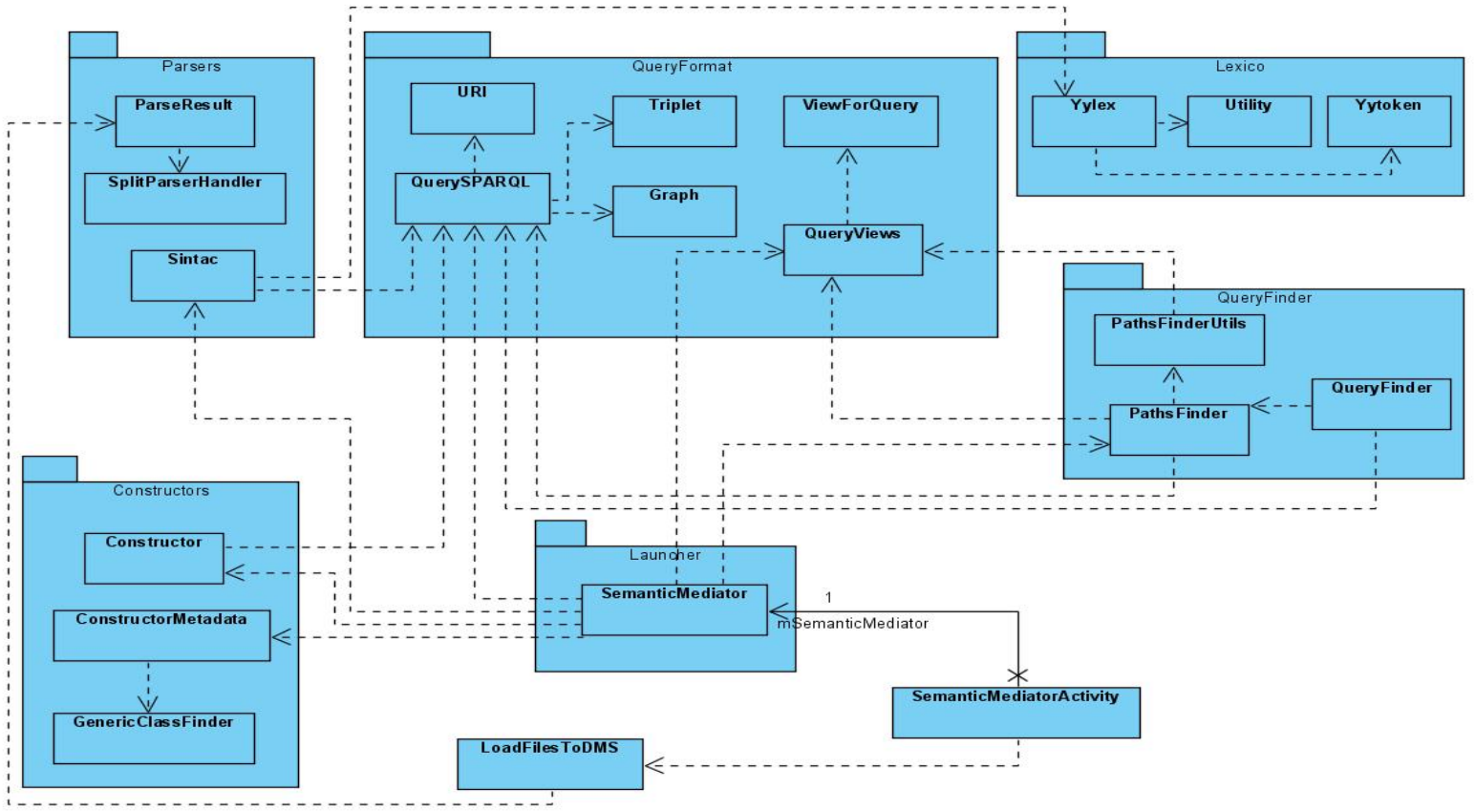

Fig 4-3 Diagrama de clases módulo *SemanticMediator*

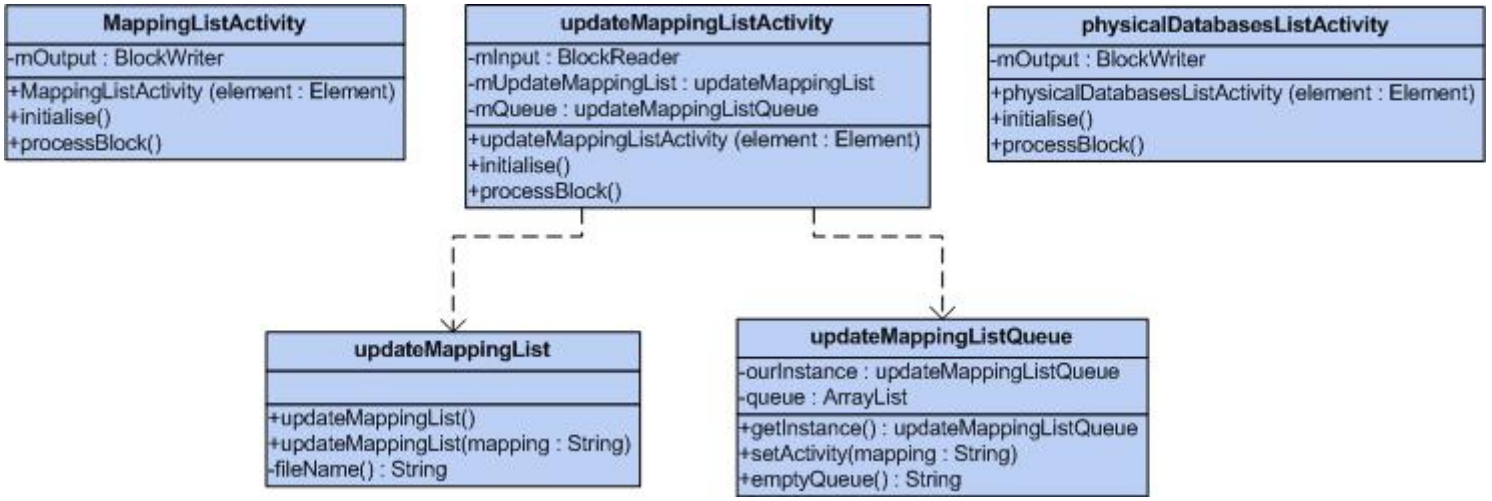

Fig 4-4 Diagrama de clases módulo *ControladorMappings*

En el diagrama de clases de la figura Fig 4-3 se muestran todas las clases del subsistema encargado de la mediación semántica en sí, en sus correspondientes paquetes o agrupaciones, además de las relaciones existentes entre ellas. No se muestran los atributos y funcionalidades de las clases por cuestiones de espacio.

En el diagrama de clases de la figura Fig 4-4 se encuentran las clases del subsistema encargado de gestionar los *mappings*. Entre las distintas clases se reparten las tareas de actualizar, listar o detallar las clases, como se expondrá posteriormente.

4.5 DESCRIPCIÓN DE LAS CLASES SOFTWARE

En esta sección se describen las clases tanto del subsistema Mediador Semántico como del subsistema Controlador de *Mappings*.

4.5.1 Mediador Semántico

A continuación se describen cada una de las clases que se han considerado más relevantes del módulo encargado de realizar la mediación semántica en el sistema.

Para empezar, la clase *SemanticMediatorActivity* (figura Fig 4-5), hereda de la clase *Activity* de OGSA-DAI (http://www.ogsadai.org.uk). Recibe las llamadas al servicio Grid y se encarga de dirigir el flujo de ejecución hacia conocer los resultados de una consulta o los identificadores de los ficheros en el repositorio DMS. Esta clase sigue el modelo de patrón Controlador.

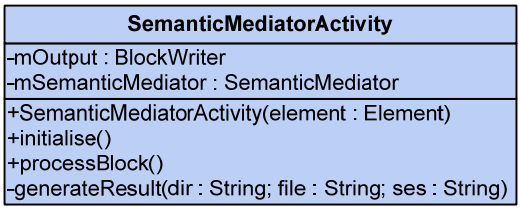

Fig 4-5 Clase SemanticMediatorActivity

La clase *SemanticMediatorActivity* llama siempre a la clase *SemantiMediator*, la cual sigue un modelo de patrón Experto. Encapsula toda la información necesaria en cada momento y se encarga de llamar al resto de clases dependiendo de la tarea a ejecutar en cada instante.

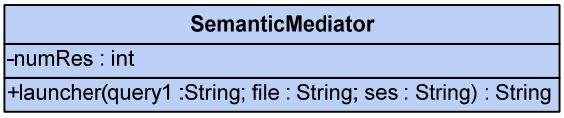

Fig 4-6 Clase SemanticMediator

Si la ejecución de este módulo comprende la actualización de ficheros en el repositorio DMS (*Data Management System*), entonces entrarán en juego los métodos de la clase *LoadFilesToDMS,* la cual subirá los ficheros (los resultados y los metadatos sobre los resultados) al repositorio integrado y recogerá la información sobre los identificadores que han sido asignados a dichos ficheros en el repositorio. Esta clase utiliza otro servicio Grid encargado de esa actualización de los ficheros de resultados y metadatos en el repositorio DMS.

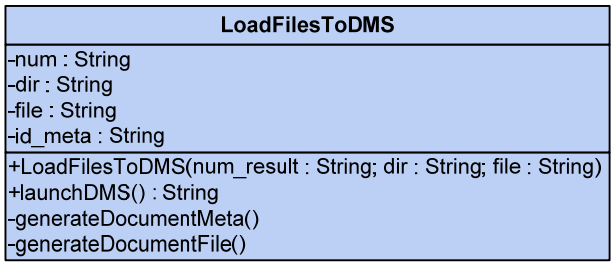

Fig 4-7 Clase LoadFilesToDMS

A continuación se detallan las clases involucradas en el proceso de mediación:

La clase *Sintac* (figura Fig 4-8), es el analizador sintáctico de una consulta en lenguaje SPARQL. Principalmente, parsea dicha consulta y la almacena en un objeto *QuerySPARQL* (figura Fig 4-12)

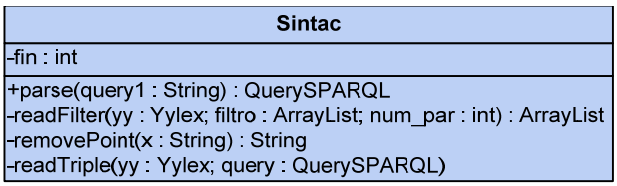

Fig 4-8 Clase Sintac

QuerySPARQL (figura Fig 4-12) formaliza la representación de una consulta en lenguaje SPARQL. Almacena los campos considerados más importantes dentro de la consulta, valiéndose para ello de otra serie de clases que formalizan la representación de una tripleta (dominio-relación-rango), de una URI, o de una vista o conjunto de vistas de una ontología. Esas clases se muestran a continuación:

- *Triplet* (figura Fig 4-10) formaliza la representación de una tripleta (dominio-relación-rango) de una consulta.

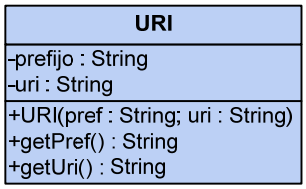

Fig 4-9 Clase URI

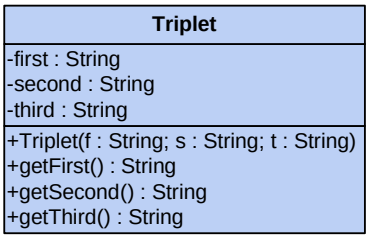

Fig 4-10 Clase Triplet

- *URI* (figura Fig 4-9), como su propio nombre indica, formaliza la representación de una URI dentro de una consulta formulada en lenguaje SPARQL.
- *QueryViews* (figura Fig 4-11), representa todas las vistas de una consulta general.

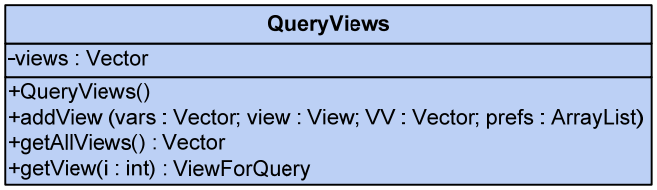

Fig 4-11 Clase QueryViews

- *ViewForQuery* (figura Fig 4-13) se encarga de formalizar el conjunto de vistas de la ontología de cada una de las consultas en las que se dividirá la consulta original. Almacena las variables que conciernen a esas vistas, las vistas en

términos de la ontología maestra y en términos físicos y una serie de vectores de cambios para realizar las transformaciones pertinentes.

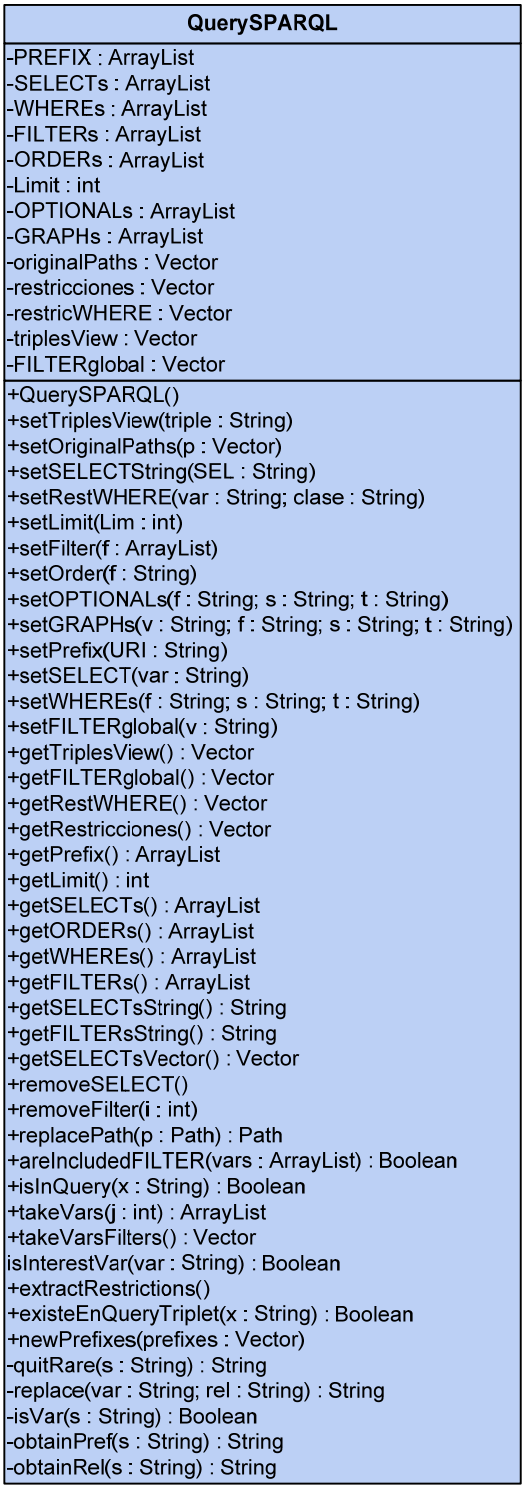

Fig 4-12 Clase QuerySPARQL

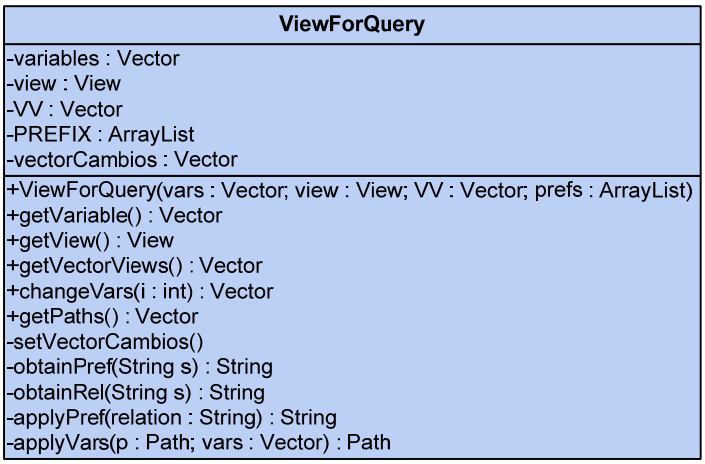

Fig 4-13 Clase ViewForQuery

Hasta aquí las clases que se encargan de formalizar la representación de una consulta en SPARQL.

La clase *QueryFinder* (figura Fig 4-14) se encarga de localizar cada una de las subconsultas de una consulta en relación a los distintos *mappings* que conoce el sistema. Esta clase gestionará las distintas vistas de cada subconsulta.

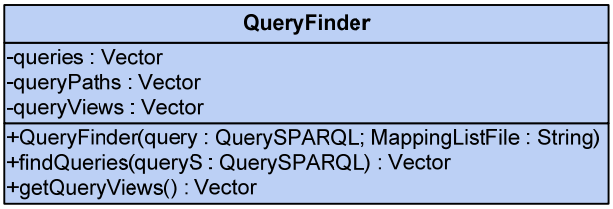

Fig 4-14 Clase QueryFinder

Por otra parte, la clase *PathsFinderUtils* (figura Fig 4-15) y *PathsFinder* (figura Fig 4-16) contienen una serie de funcionalidades que dan soporte a la localización de las vistas de la ontología de una consulta en distintos mappings y a su posterior traducción y almacenaje en cada subconsulta.

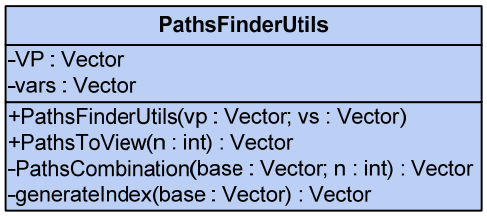

Fig 4-15 Clase PathsFinderUtils

Fig 4-16 Clase PathsFinder

La clase *Constructor* (figura Fig 4-17), se encarga de transformar a *String* una consulta en formato *QuerySPARQL*.

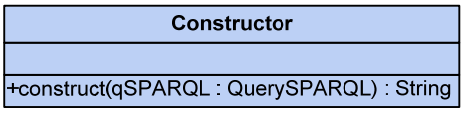

Fig 4-17 Clase Constructor

Las clases *ConstructorMetadata*, *ParseResult*, *GenericClassFinder* y *SplitParserHandler* (figuras Fig 4-18, Fig 4-19, Fig 4-20 y Fig 4-21, respectivamente), contienen las funcionalidades necesarias para obtener los metadatos de una consulta y sus resultados y presentarlos en un fichero con el formato adecuado. En el caso de que los resultados a una consulta deseen almacenarse en el repositorio integrado DMS, entonces deben generarse también los metadatos sobre los datos resultado y almacenarse en el mismo directorio que éstos.

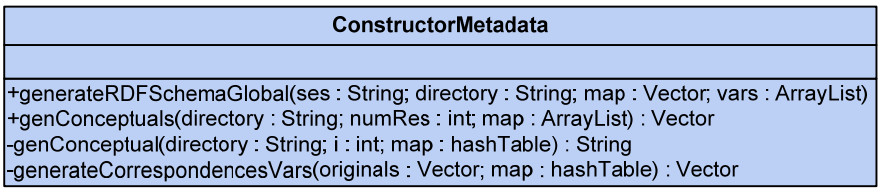

Fig 4-18 Clase ConstructorMetadata

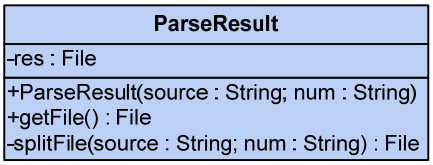

Fig 4-19 Clase ParseResult

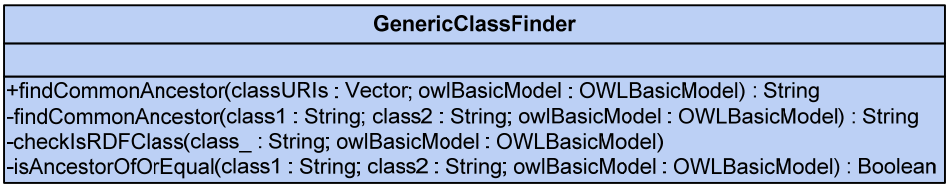

Fig 4-20 Clase GenericClassFinder

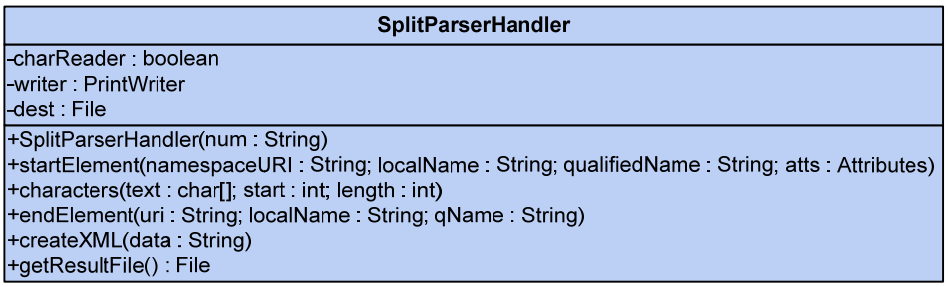

Fig 4-21 Clase SplitParserHandler

4.5.2 Módulo Controlador de *Mappings*

En este módulo se encuentran las clases que implementan los servicios Grid del sistema para actualizar, listar y detallar los *mappings* existentes en el mismo, además de alguna otra clase de apoyo al subsistema.

En el servicio Grid de listado de mappings disponibles en el sistema existe una única clase (figura Fig 4-22). La clase *MappingListActivity* hereda de la clase *Activity* de OGSA-DAI (http://www.ogsadai.org.uk). Es una clase bastante sencilla cuya funcionalidad consiste en acceder a un fichero del sistema donde se listan los *mappings* y leer los *mappings* que el sistema conoce.

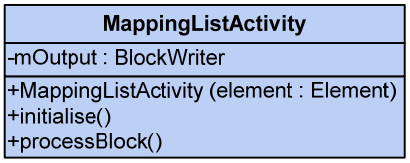

Fig 4-22 Clase MappingListActivity

En el servicio Grid que se encarga de detallar los *mappings* disponibles, existe también una única clase (figura Fig 4-23). Su funcionamiento en bastante parecido a la clase anterior. Hereda también de la clase *Activity* de OGSA-DAI, pero en este caso no se limita a listar los nombres de los *mappings* disponibles en el sistema, sino que localiza cada uno de ellos y lee el contenido de los mismos.

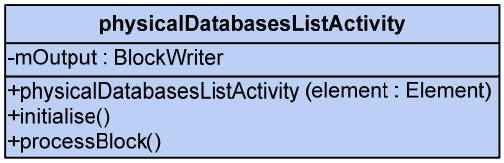

Fig 4-23 Clase physicalDatabasesListActivity

El último servicio Grid del módulo controlador de *Mappings*, es el encargado de actualizar un *mapping* en el sistema. Procede de forma que cuando es llamado, la llamada se inserta en una cola de espera (figura Fig 4-24), para evitar la actualización de *mappings* a la vez.

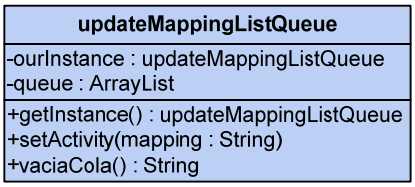

Fig 4-24 Clase updateMappingListQueue

El servicio de actualización de *mappings* (figura Fig 4-25) es un servicio OGSA-DAI también y por tanto hereda de la clase *Activity* de OGSA-DAI. Este servicio, además de insertar la llamada en la cola de espera, coge todos los elementos de la misma y, secuencialmente, los actualiza en el sistema.

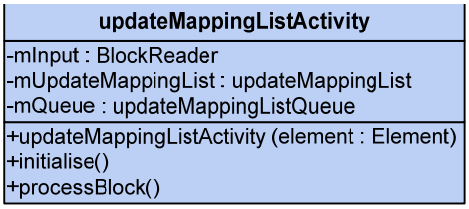

Fig 4-25 Clase updateMappingListActivity

El servicio en cuestión, haciendo uso de la clase *updateMappingList* (figura Fig 4-26), comprueba si el *mapping* es nuevo en el sistema o si es una modificación de alguno ya existente.

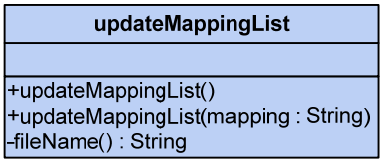

Fig 4-26 Clase updateMappingList

5 EXPERIMENTOS Y RESULTADOS

5.1 INTRODUCCIÓN

La solución propuesta pretende ser usada dentro del entorno del proyecto ACGT.

En este capítulo se describen los experimentos realizados con la solución propuesta para comprobar el buen funcionamiento del sistema implementado en este proyecto. El objetivo es comprobar que con las distintas herramientas desarrolladas, la capa de mediación del entorno ACGT (en concreto, el Mediador Semántico, y los servicios de actualización y consulta de *mappings*) funcionan correctamente.

En la siguiente tabla se muestra un esquema de cada uno de los experimentos realizados.

Servicio	Casos	Descripción	Sección
"MappingListActivity"	No existen *mappings* en el sistema	El sistema devuelve que el número de *mappings* es 0.	5.2.1.
	Existen *mappings* en el sistema	El sistema devuelve el número de *mappings* disponibles y los lista.	5.2.2.
"physicalDatabasesList"	No se conocen *wrappers*	El sistema devuelve una lista vacía.	5.3.1.
	Se conocen *wrappers*	El sistema devuelve una lista con los *wrappers* disponibles.	5.3.2.
"updateMappingListActivity"	Crear nuevo mapping	El sistema creará y modificará los archivos que sean necesarios.	5.4.
	Modificar mapping	El sistema modificará los archivos que sean necesarios.	5.4.
Creación de esquema .n3 de Base de Datos relacional	-	El sistema creará un esquema .n3 de la Base de	5.5.

		Datos especificada.	
"SemanticMediator"	Actualización del fichero de resultados en el repositorio integrado	El sistema realiza la consulta y a continuación actualiza el repositorio integrado con los resultados de la misma.	5.6.
	Respuesta a consulta SPARQL	El sistema responde a la consulta realizada y realiza la integración de los resultados.	5.6.1., 5.6.2., 5.6.3.

5.2 EXPERIMENTOS "MAPPINGLISTACTIVITY"

Este servicio devuelve la lista de los *mappings* disponibles en el mediador, es decir, responde al caso de uso "Listar *Mappings*". Para probar este servicio se han realizado pruebas sobre dos casos opuestos, la existencia o no de *mappings* en el sistema.

La llamada, en cualquiera de los dos casos, que se debe realizar al sistema, tiene la siguiente forma:

```
  C:\ogsadai-wsrf> ant dataServiceClient
  -Ddai.url=
  http://servet.dia.fi.upm.es:8080/wsrf/services/ogsadai/DataSer
vice
  -Ddai.resource.id=MappingListActivity
  -Ddai.action=fichero_consulta.xml
```

Donde fichero_consulta.xml tiene el siguiente contenido: (ver figura Fig 5-1)

```
<?xml version="1.0" encoding="UTF-8" ?>
- <perform xmlns="http://ogsadai.org.uk/namespaces/2005/10/types">
    <!--  create session to allow notification of asynchronous delivery completion  -->
    <session timeout="300000" />
  - <MappingList name="_MappingList_">
      <MappingListOutput>result</MappingListOutput>
    </MappingList>
  </perform>
```

Fig 5-1 Contenido "fichero_consulta.xml"

5.2.1 "MappingListActivity": No existen mappings

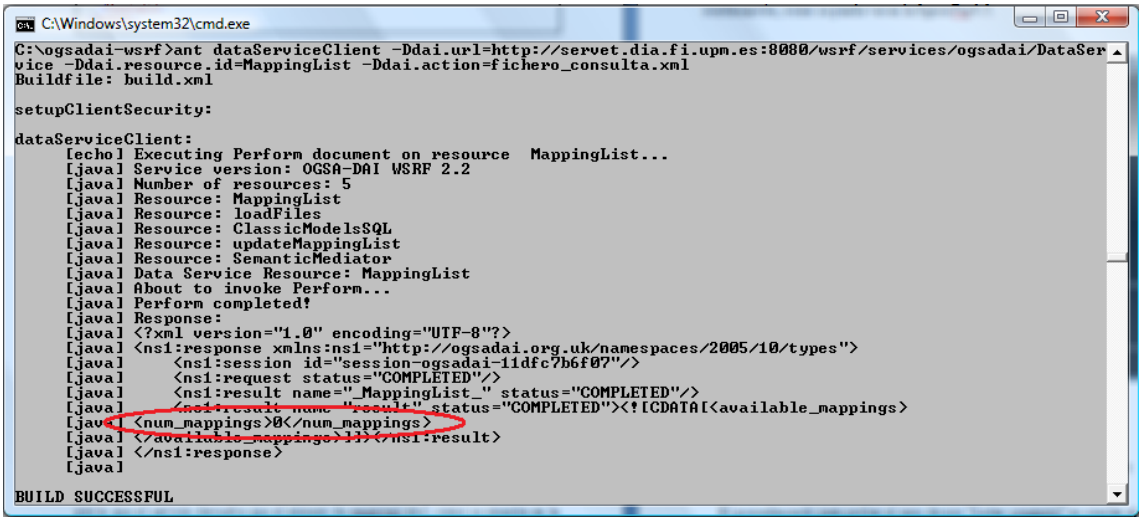

Fig 5-2 "MappingListActivity": no existen *mappings* en el sistema

En el primer experimento sobre el servicio, no se conocen *mappings* en el sistema, por lo que el servicio devuelve que el número de *mappings* es 0, como se muestra en la figura Fig 5-2.

5.2.2 “MappingListActivity”: Existen mappings

En este experimento, por el contrario, existen dos *mappings* en el sistema. El servicio así lo corrobora, diciendo que el número de *mappings* es 2 y mostrándolos a continuación, como se puede ver en la figura Fig 5-3.

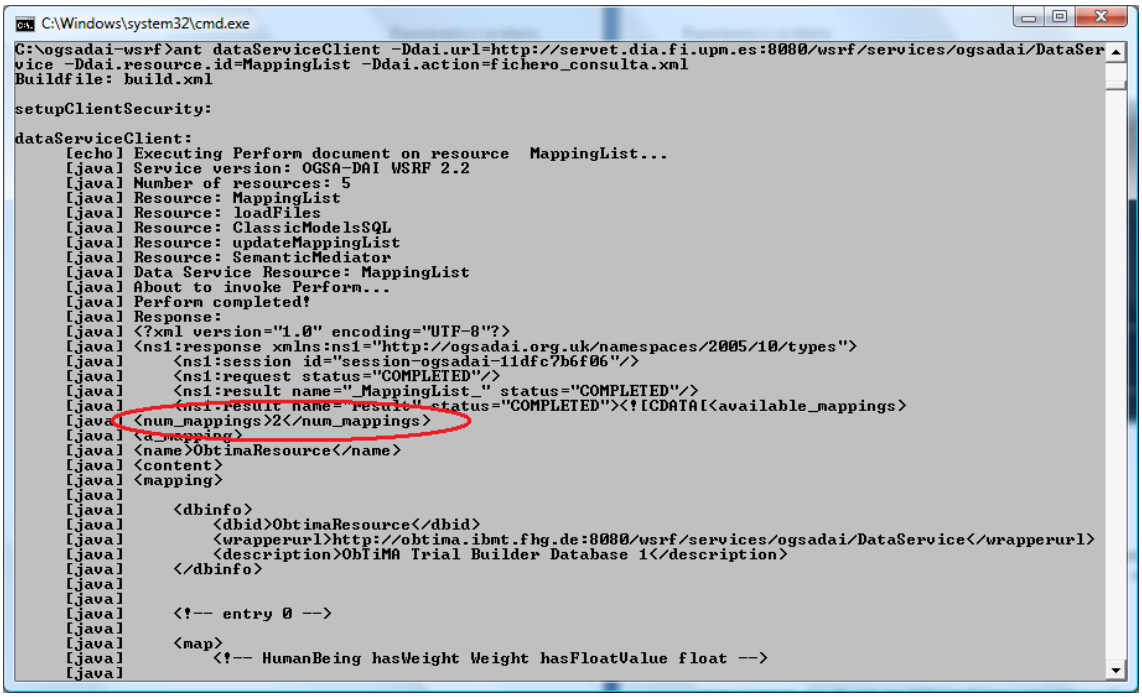

Fig 5-3 “MappingListActivity”: existen *mappings* en el sistema

5.3 EXPERIMENTOS “PHYSICALDATABASESLIST”

El procedimiento para probar el caso de uso “Listar *wrappers*” es similar al anterior. Se han realizado dos pruebas sobre dos casos opuestos: el conocimiento o no de *wrappers* en el sistema.

La llamada a realizar en este caso es:

```
C:\ogsadai-wsrf> ant dataServiceClient
-Ddai.url=
http://servet.dia.fi.upm.es:8080/wsrf/services/ogsadai/DataSer
vice
-Ddai.resource.id=physicalDatabasesList
-Ddai.action=fichero_consulta.xml
```

Donde fichero_consulta.xml tiene el siguiente contenido: (ver figura Fig 5-4)

```
<?xml version="1.0" encoding="UTF-8" ?>
- <perform xmlns="http://ogsadai.org.uk/namespaces/2005/10/types">
    <!--  create session to allow notification of asynchronous delivery completion  -->
    <session timeout="300000" />
  - <physicalDatabasesList name="procesando_PhysicalDatabasesList">
      <physicalDatabasesListOutput>resultados</physicalDatabasesListOutput>
    </physicalDatabasesList>
  </perform>
```

Fig 5-4 Contenido “fichero_consulta.xml”

5.3.1 “physicalDatabasesList”: no se conocen wrappers

La consulta al sistema se prueba en el mismo caso que para el experimento 1. No existen *mappings* en el sistema por lo que no se conocen *wrappers*. Este comportamiento se muestra en la figura Fig 5-5.

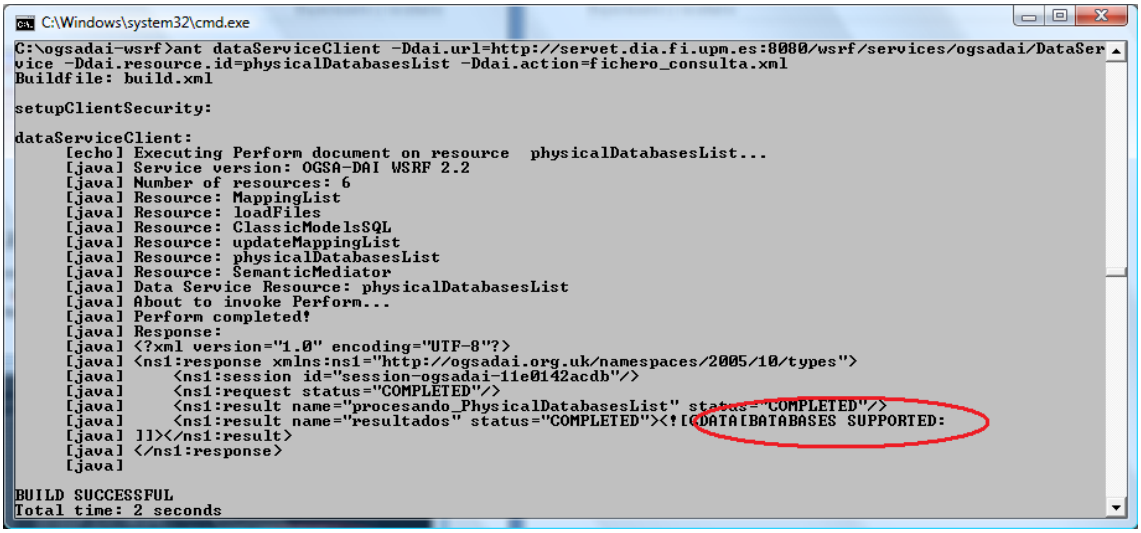

Fig 5-5 “physicalDatabasesList”: el sistema no conoce *wrappers*

5.3.2 “physicalDatabasesList”: se conocen wrappers

La consulta al sistema se prueba en el mismo caso que para el experimento 2. Existen *mappings* en el sistema por lo que sí se conocen *wrappers* (a los que referencien dichos *mappings*). Este comportamiento se muestra en la figura Fig 5-6, en la que se citan los *wrappers* que el sistema conoce.

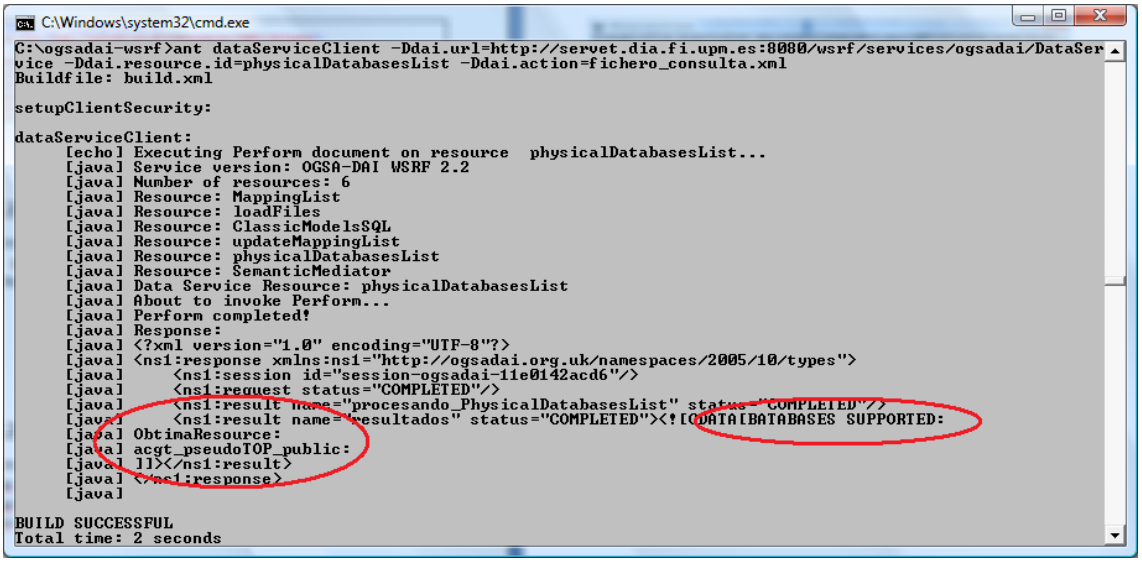

Fig 5-6 “physicalDatabasesList”: el sistema conoce *wrappers*

5.4 EXPERIMENTOS “UPDATEMAPPINGLISTACTIVITY”

Dentro del caso de uso “Actualizar mapping”, que es al que responde el servicio “updateMappingListActivity”, existen dos casos claramente diferenciados: la actualización puede darse sobre un *mapping* ya existente o puede crearse un *mapping* nuevo. La llamada al sistema es la misma en ambos casos, y será el sistema el que deba diferenciar entre uno u otro, llevando a cabo una acción de actualización o de creación en cada caso.

La llamada genérica al servicio sería:

```
  C:\ogsadai-wsrf> ant dataServiceClient
  -Ddai.url=
  http://servet.dia.fi.upm.es:8080/wsrf/services/ogsadai/DataS
ervice
  -Ddai.resource.id=updateMappingList
  -Ddai.action=fichero_mapping.xml
```

Donde fichero_mapping.xml contendría el *mapping* a actualizar/crear con el formato de la figura Fig 5-7 (sustituyendo "..mapping.." por el *mapping* en sí).

```
<?xml version="1.0" encoding="UTF-8" ?>
- <perform xmlns="http://ogsadai.org.uk/namespaces/2005/10/types">
    <!--  create session to allow notification of asynchronous delivery completion  -->
    <session timeout="300000" />
    <!--  retrieval of images belonging to the specified (unique) patient ids  -->
  - <updateMappingList name="UMList">
      <updateMappingListInput>..mapping..</updateMappingListInput>
    </updateMappingList>
  </perform>
```

Fig 5-7 Contenido "fichero_mapping.xml"

Se realizaron dos pruebas, una para la creación y otra para la actualización de un fichero de mapping. El resultado en ambos casos es correcto, y las modificaciones en el sistema en cada caso se muestran en la tabla de la figura Fig 5-8. (NOTA: el fichero MappingListFile.xml es el que contiene las rutas a todos los *mappings* conocidos en el sistema)

	Modificación de ficheros	**Creación de ficheros**
"ACTUALIZAR"	Modificación del fichero que contenía el *mapping* sustituyéndolo por el nuevo Modificación del fichero MappingListFile.xml	-
"CREAR"	Modificación del fichero MappingListFile.xml	Creación de un nuevo fichero que contenga el *mapping*

Fig 5-8 Modificaciones en el sistema en "updateMappingList"

5.5 CREACIÓN DEL ESQUEMA .N3 DE UNA BASE DE DATOS RELACIONAL

Este caso de uso ha sido especialmente útil para realizar pruebas en el mediador semántico y para crear *wrappers* que envuelvan Bases de Datos relacionales locales. Las pruebas que se han realizado para el mismo derivan de las necesidades que se han tenido y su funcionamiento.

Para la creación del esquema .n3 de una Base de Datos es necesario proporcionar:

- El driver necesario del lenguaje de la Base de Datos (en el caso de ente trabajo de Fin de Carrera se ha realizado exclusivamente con un driver para mySQL).
- El nombre de usuario y la contraseña necesarios para acceder a la Base de Datos (si lo hubiera).
- La localización de la Base de Datos de la que se quiere crear el esquema.

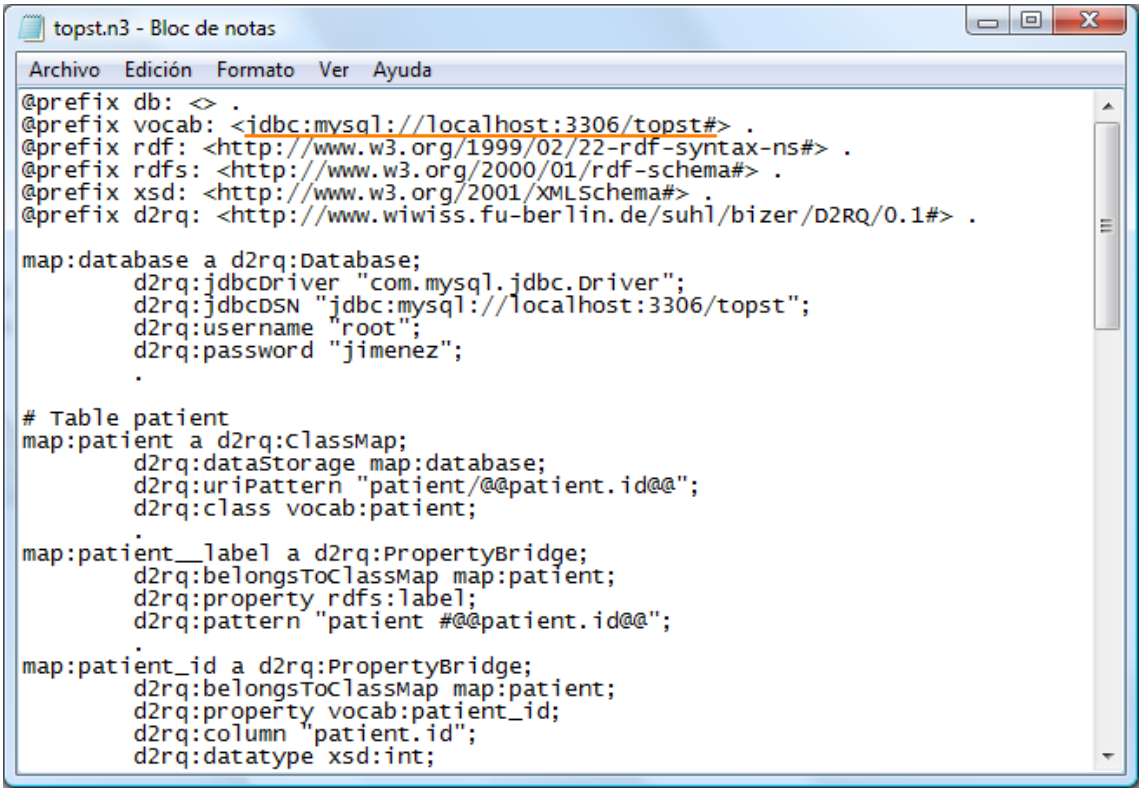

Fig 5-9 Esquema .n3

Un ejemplo del esquema creado se muestra en la figura Fig 5-9. Será necesario rellenar manualmente el valor de “vocab” con la localización de la Base de Datos, ya que el programa en cuestión no lo hace.

5.6 EXPERIMENTOS “SEMANTICMEDIATOR”

El mediador semántico se ha probado de tres formas distintas:

- No hay *mappings* en el sistema, y por tanto el mediador no conoce los *wrappers* que envuelven a las distintas Bases de Datos.
- Existe un único *mapping*. Con esta prueba se verifica que el mediador semántico accede correctamente a los *wrappers*.
- Existen varios *mappings* relacionados con distintas fuentes de datos heterogéneas. En este caso, además de comprobar que el mediador accede correctamente a los *wrappers*, se debe verificar que la integración de los resultados funciona correctamente.

Además se ha probado que el mediador actualice correctamente los resultados en el repositorio DMS.

La llamada genérica al servicio, en cualquier caso sería:

```
C:\ogsadai-wsrf> ant dataServiceClient
-Ddai.url=
http://servet.dia.fi.upm.es:8080/wsrf/services/ogsadai/DataSer
vice
-Ddai.resource.id=SemanticMediator
-Ddai.action=fichero_consulta.xml
```

Donde fichero_consulta.xml contendría la consulta que se quiere lanzar contra el sistema además del tipo de resultado que se desea obtener (o bien los resultados en un fichero .csv o los identificadores de los resultados en el repositorio DMS).

El formato de fichero_consulta.xml para el caso de devolución de los resultados en un fichero .csv sería el de la figura Fig 5-10.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <perform xmlns="http://ogsadai.org.uk/namespaces/2005/10/types">
    <documentation>Simple query across all tables in the database.</documentation>
  - <SemanticMediatorWIP name="myActivityInstance">
      <query>PREFIX acgt: <http://www.ifomis.org/acgt/1.0#> PREFIX xsd:
        <http://www.w3.org/2001/XMLSchema#> PREFIX bfo_span:
        <http://www.ifomis.org/bfo/1.1/span#> PREFIX bfo_snap:
        <http://www.ifomis.org/bfo/1.1/snap#> PREFIX ro:
        <http://www.ifomis.org/obo/ro/1.0#> SELECT ?patientID ?birthDate ?regDate
        WHERE { ?patient acgt:hasIdentifier ?patientID . ?patient a acgt:HumanBeing . ?
        patientID a xsd:string . ?patient2 acgt:hasBirthDate ?birthDate . ?patient2 a
        acgt:HumanBeing . ?birthDate a xsd:date . ?registration acgt:hasPatient ?
        patient3 . ?registration a acgt:Registration . ?patient3 a acgt:HumanBeing . ?
        registration2 acgt:hasProcessEnd ?endOfReg . ?endOfReg acgt:hasDate ?regDate . ?
        registration2 a acgt:Registration . ?endOfReg a bfo_span:ProcessBoundary . ?
        regDate a xsd:date . FILTER ( ?patient = ?patient2 ) FILTER ( ?patient = ?patient3 )
        FILTER ( ?registration = ?registration2 ) } LIMIT 5</query>
      <type>CSV</type>
      <SemanticMediatorOutput name="myResults" />
    </SemanticMediatorWIP>
  </perform>
```

Fig 5-10 Formato del fichero_consulta.xml (tipo CSV)

La consulta que recibe el mediador semántico está recogida dentro de la etiqueta SemanticMediatorWIP. Dentro de ella existen distintos campos:

- query: la consulta en lenguaje SPARQL al Mediador Semántico.
- type: el tipo de consulta que se desea realizar. Puesto que en este caso sólo se desean conocer los resultados en formato .csv, ese campo deberá rellanarse con las letras “CSV”.
- SemanticMediatorOutput: es la etiqueta de la que “colgarán” los resultados cuando éstos sean devueltos.

Un resultado típico a este tipo de consulta sería el de la figura Fig 5-11. En dicha figura se observa un formato de tabla para los resultados, donde cada tupla de los mismos está separada por comas.

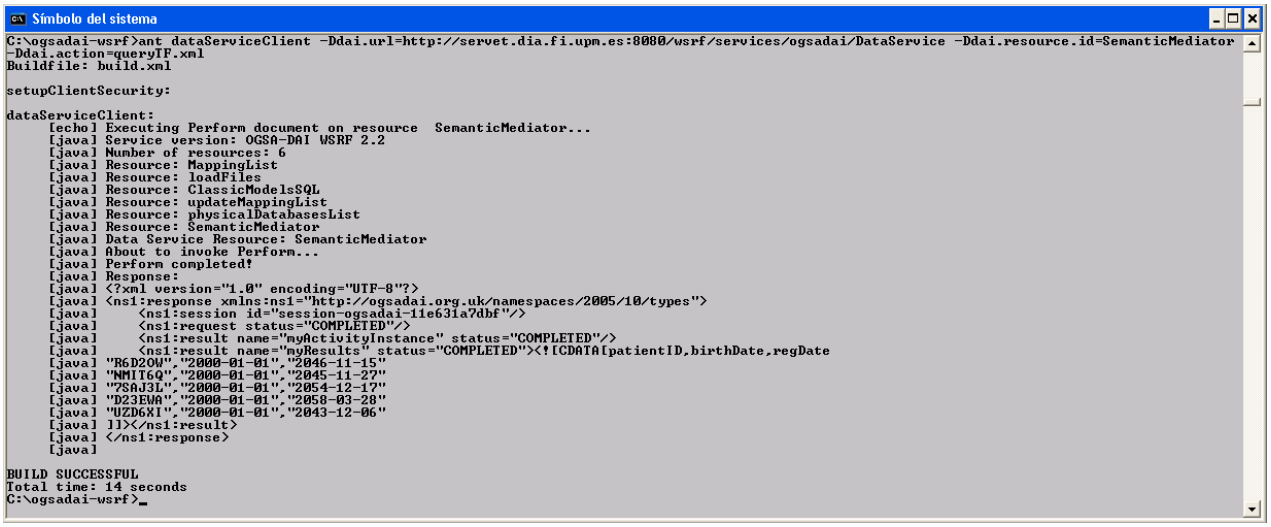

Fig 5-11 Resultado del Mediador Semántico a una consulta tipo CSV

El formato del fichero_consulta.xml para el caso de que se desearan actualizar los resultados en el repositorio DMS junto con los metadatos asociados sería el de la figura Fig 5-12.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <perform xmlns="http://ogsadai.org.uk/namespaces/2005/10/types">
    <documentation>Simple query across all tables in the database.</documentation>
  - <SemanticMediatorWIP name="myActivityInstance">
      <query>PREFIX acgt: <http://www.ifomis.org/acgt/1.0#> PREFIX xsd:
        <http://www.w3.org/2001/XMLSchema#> PREFIX bfo_span:
        <http://www.ifomis.org/bfo/1.1/span#> PREFIX bfo_snap:
        <http://www.ifomis.org/bfo/1.1/snap#> PREFIX ro: <http://www.ifomis.org/obo/ro/1.0#>
        SELECT ?patientID ?widthValue ?heightValue ?pcrYN WHERE { ?patient acgt:hasIdentifier ?
        patientID . ?patient a acgt:HumanBeing . ?patientID a xsd:string . ?patient2 ro:hasPart ?
        neoplasm . ?neoplasm acgt:undergoes ?diagnosticProcess . ?patient2 a acgt:HumanBeing . ?
        neoplasm a acgt:Neoplasm . ?diagnosticProcess a acgt:DiagnosticProcess . ?diagnosticProcess2
        acgt:reveals ?width . ?width acgt:hasFloatValue ?widthValue . ?diagnosticProcess2 a
        acgt:DiagnosticProcess . ?width a acgt:Width . ?widthValue a xsd:float . ?diagnosticProcess3
        acgt:reveals ?height . ?height acgt:hasFloatValue ?heightValue . ?diagnosticProcess3 a
        acgt:DiagnosticProcess . ?height a acgt:Height . ?heightValue a xsd:float . ?diagnosticProcess4
        acgt:reveals ?pcrStatus . ?pcrStatus acgt:hasBooleanValue ?pcrYN . ?diagnosticProcess4 a
        acgt:DiagnosticProcess . ?pcrStatus a acgt:pCRStatus . ?pcrYN a xsd:boolean . ?chemotherapy
        ro:precedes ?diagnosticProcess5 . ?chemotherapy a acgt:Chemotherapy . ?diagnosticProcess5
        a acgt:DiagnosticProcess . FILTER ( ?patient = ?patient2 ) FILTER ( ?diagnosticProcess = ?
        diagnosticProcess2 ) FILTER ( ?diagnosticProcess = ?diagnosticProcess3 ) FILTER ( ?
        diagnosticProcess = ?diagnosticProcess4 ) FILTER ( ?diagnosticProcess = ?
        diagnosticProcess5 ) }</query>
      <type>DMS</type>
      <dir>MediatorResults</dir>
      <file>query5.csv</file>
      <SemanticMediatorOutput name="sparqlResults" />
    </SemanticMediatorWIP>
  </perform>
```

Fig 5-12 Formato del fichero_consulta.xml (tipo DMS)

En este caso, la consulta que recibe el mediador semántico está recogida también dentro de la etiqueta SemanticMediatorWIP. Sin embargo, existen diferencias en los campos internos:

- query: la consulta en lenguaje SPARQL al Mediador Semántico.
- type: el tipo de consulta que se desea realizar. En este caso se desean conocer los identificadores que recibirán tanto la carpeta de destino como el fichero con los resultados, en el repositorio DMS. Por ello, este campo deberá rellenarse con las letras “DMS”.
- dir: esta etiqueta contiene el nombre del directorio en el que se almacenará el fichero con los resultados.
- file: esta etiqueta almacena el nombre del fichero que contendrá los resultados a la consulta en formato .csv.
- SemanticMediatorOutput: es la etiqueta de la que “colgarán” los resultados cuando éstos sean devueltos.

Ahora no se devuelven los resultados explícitamente, sino que se actualizan en un repositorio integrado y se devuelven los identificadores del directorio y del fichero con los resultados en el repositorio (figura Fig 5-13).

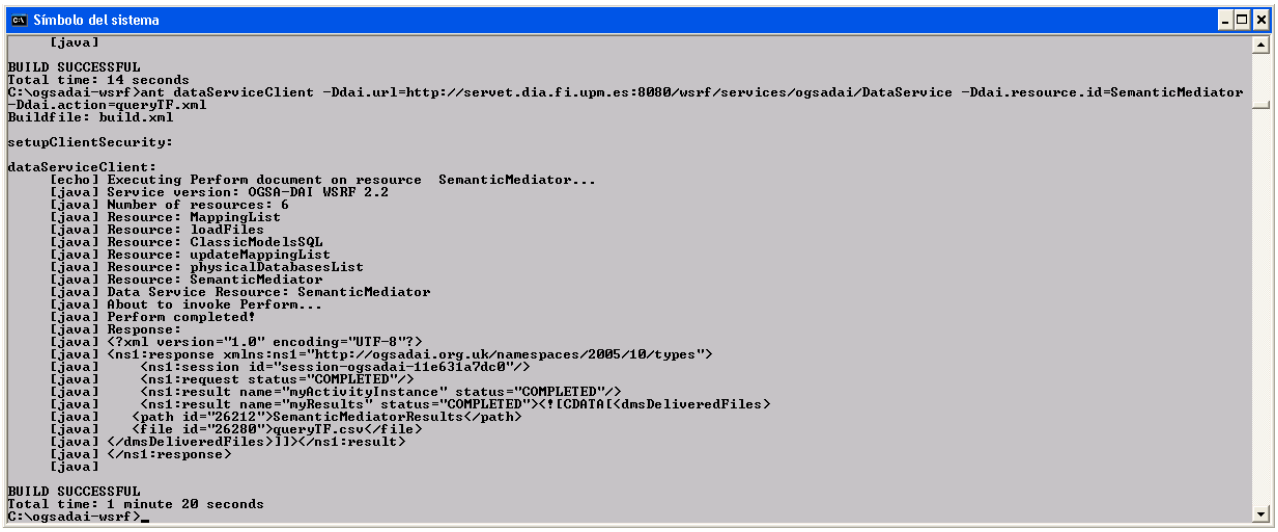

Fig 5-13 Resultado del Mediador Semántico a una consulta tipo DMS

A continuación se explica cada uno de esos identificadores devueltos así como la creación en el repositorio de los distintos archivos y directorios para el almacenaje de los resultados del mediador semántico.

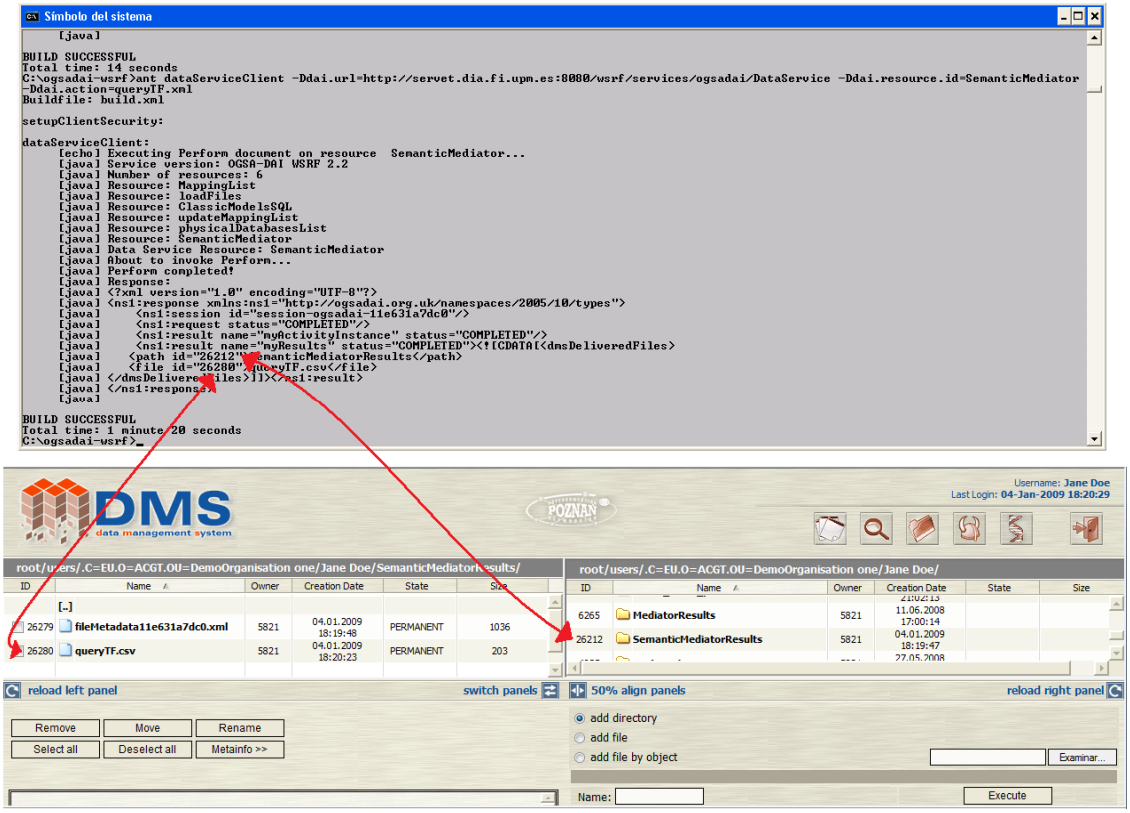

Fig 5-14 Actualización en el repositorio

Como se ha comentado, los resultados devueltos por la llamada al mediador semántico son dos identificadores, correspondientes al directorio donde se almacenarán los resultados (que se creará en caso de que no exista) y el fichero que contiene los resultados en formato .csv, tal y como se muestra en la figura Fig 5-14.

```
<?xml version="1.0" ?>
- <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:vocab_0="http://www.ifomis.org/bfo/1.1/span#" xmlns:xsd="http://www.w3.org/2001/XMLSchema#" xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://semantic_mediator_metadata#" xml:base="http://semantic_mediator_metadata">
  - <owl:Class rdf:about="http://www.ifomis.org/bfo/1.1/span#ProcessBoundary">
    - <rdfs:subClassOf>
        <owl:Class rdf:about="" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:about="http://www.w3.org/2001/XMLSchema#string">
    - <rdfs:subClassOf>
        <owl:Class rdf:about="" />
      </rdfs:subClassOf>
    </owl:Class>
  - <owl:Class rdf:about="http://www.w3.org/2001/XMLSchema#date">
    - <rdfs:subClassOf>
        <owl:Class rdf:about="" />
      </rdfs:subClassOf>
    </owl:Class>
    <vocab_0:ProcessBoundary rdf:ID="regDate" />
    <xsd:date rdf:ID="birthDate" />
    <xsd:string rdf:ID="patientID" />
</rdf:RDF>
```

Fig 5-15 Ejemplo de fichero de metadatos

Además de crearse en el repositorio el directorio que contendrá los resultados se crea también un fichero con los metadatos relativos a los atributos consultados. Un ejemplo de ese fichero se muestra en la figura Fig 5-15.

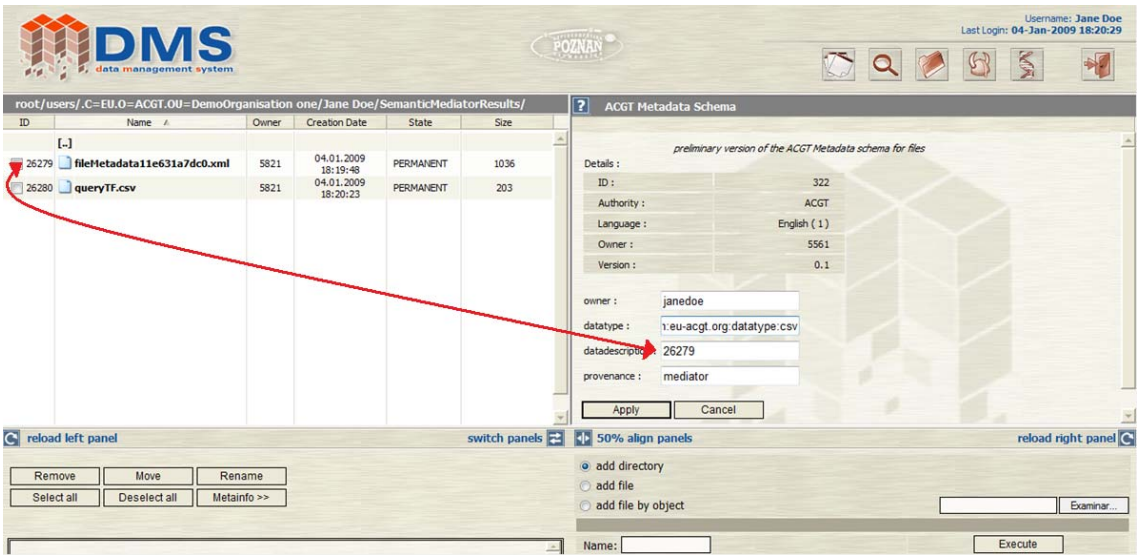

Fig 5-16 Atributos del fichero de los resultados

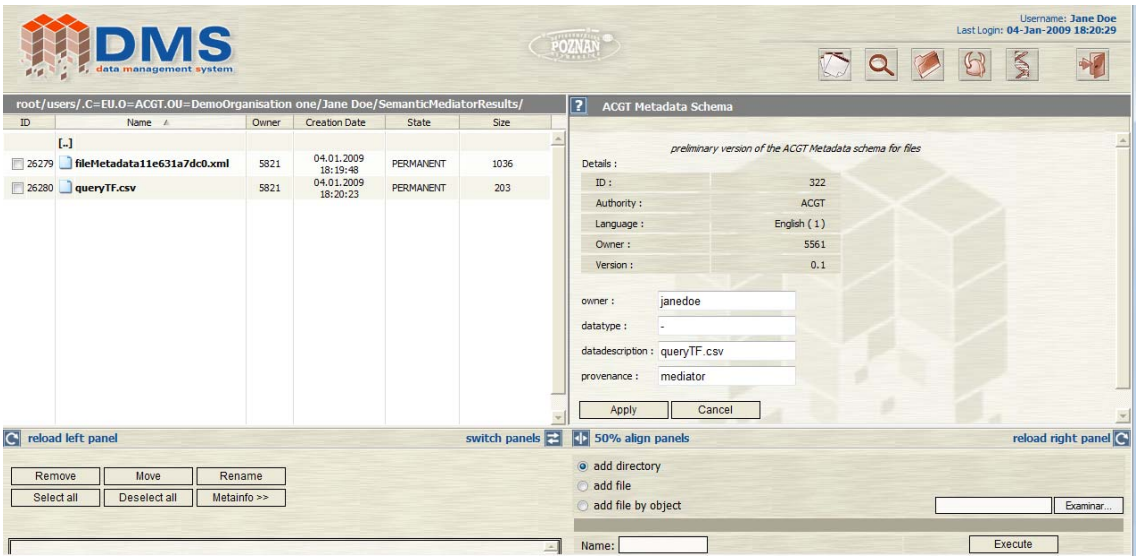

Fig 5-17 Atributos del fichero de metadatos

Al crear los ficheros de los resultados y los metadatos, se crean unos atributos asociados a ellos, como son: el propietario del fichero, el tipo de datos, la descripción de

los datos (en el caso de de los resultados su descripción se muestra en el fichero de los metadatos, como se puede ver en la figura Fig 5-16) y su origen, que en ambos casos será el mediador semántico. Estos atributos de los ficheros de resultados y metadatos se muestran en las figuras Fig 5-16 y Fig 5-17 respectivamente.

Todos los archivos se suben correctamente al repositorio, y los identificadores que devuelve la consulta son correctos, por tanto sólo queda probar que las consultas en sí acceden bien a las Bases de Datos en caso de que existan y que se integran bien los resultados.

En los siguientes apartados se mostrará cómo funciona el mediador semántico en caso de que:

- No existan *mappings*: el mediador deberá responder que no hay resultados a la consulta. El hecho de que no existan *mappings* en el sistema es transparente al usuario.
- Existe un único *mapping*: el mediador deberá responder con los resultados que encuentre en la Base de Datos asociada.
- Existen varios *mappings*: en este caso el mediador deberá consultar cada unos de los *mappings* que tenga disponibles y dividir la consulta o no si va referida a distintas Bases de Datos. Deberá lanzar cada una de las subconsultas al *wrapper* correspondiente, recoger los resultados de cada uno, integrarlos y presentarlos de forma homogénea al usuario.

En cualquier caso, el usuario no sabe si está accediendo a un *mapping*, a varios o a ninguno.

5.6.1 “SemanticMediator”: no hay *mappings*

En este experimento se lanza una consulta (la consulta de la figura Fig 5-10) contra el mediador semántico y éste no conoce ningún *mapping*. El sistema por tanto no tiene datos para responder a la consulta y no devuelve ningún resultado, como se muestra en la figura Fig 5-18.

Como se ha comentado, el hecho de que no existan *mappings* es transparente al usuario, que simplemente recibe que no hay resultados a su consulta.

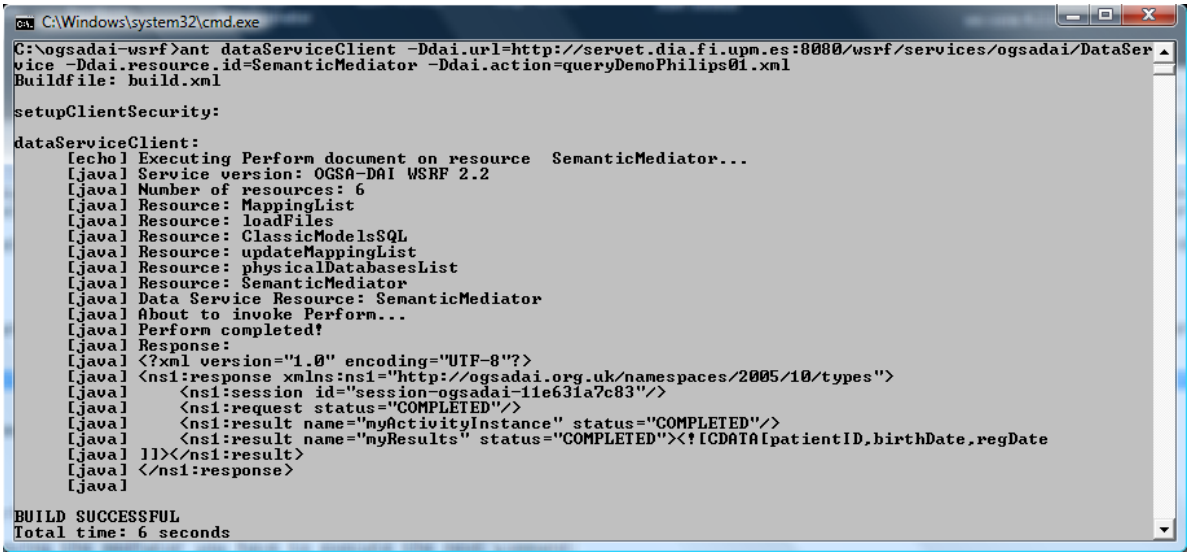

Fig 5-18 “SemanticMediator”: no existen *mappings*

5.6.2 “SemanticMediator”: existe un *mapping*

En este caso, el mediador recoge los resultados que le devuelve el único wrapper que conoce y se los presenta al usuario tal y como muestra la figura Fig 5-19.

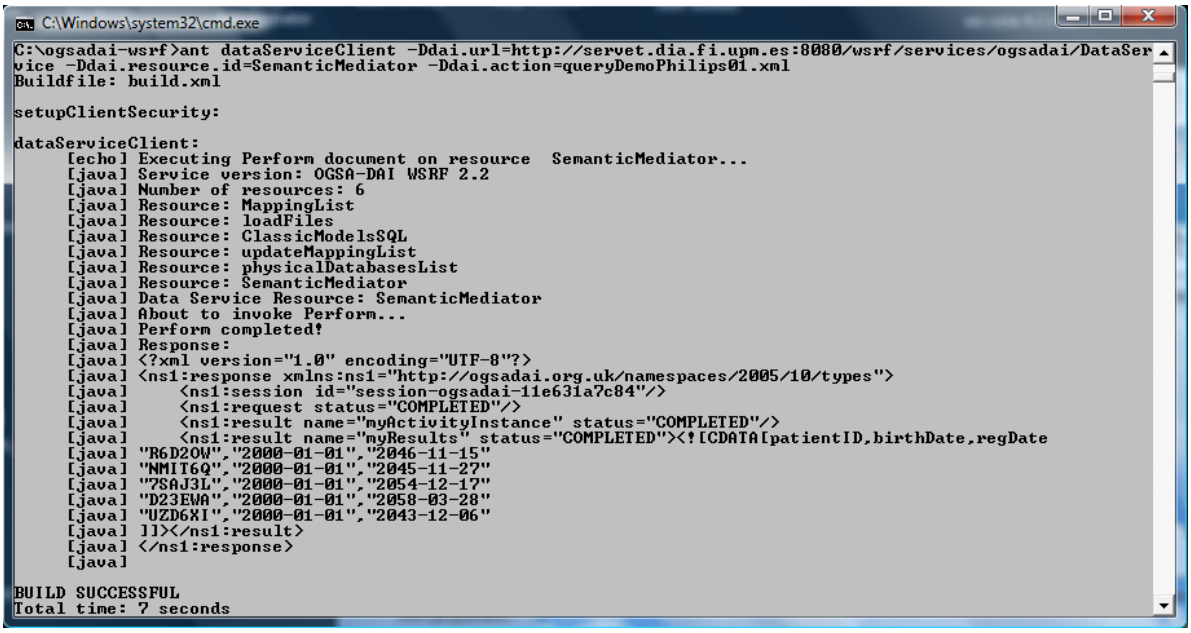

Fig 5-19 “SemanticMediator”: existe un *mapping*

5.6.3 “Semantic Mediator”: existen varios *mappings*

En este conjunto de experimentos lo realmente importante es demostrar que se resuelven los distintos casos de heterogeneidad semántica (a nivel de esquema) que pueden surgir al tener que integrar resultados que provengan de distintas fuentes de datos. Se trata de resolver las heterogeneidades que pueden surgir a nivel semántico ya que las que surgen a nivel sintáctico se ven resueltas por los *wrappers*.

Para esta serie de experimentos se van a usar tres Bases de Datos creadas para tal efecto, las cuales se muestran en la siguiente figura

BASE de DATOS “A”		
Paciente	Fecha_Ingreso	Edad
1	2008-05-04	57
2	2008-06-03	64
3	2007-10-17	50
4	2008-07-25	48
…	…	…

BASE de DATOS “B”	
ID_P	Fecha_Diagnóstico
1	2008-06-05
2	2006-05-02
6	2008-06-03
7	2008-10-22
…	…

BASE de DATOS “C”	
Ant	PrA
A	bh5
B	r56
D	ss7
C	a1
…	…

Fig 5-20 Bases de Datos usadas en los ejemplos

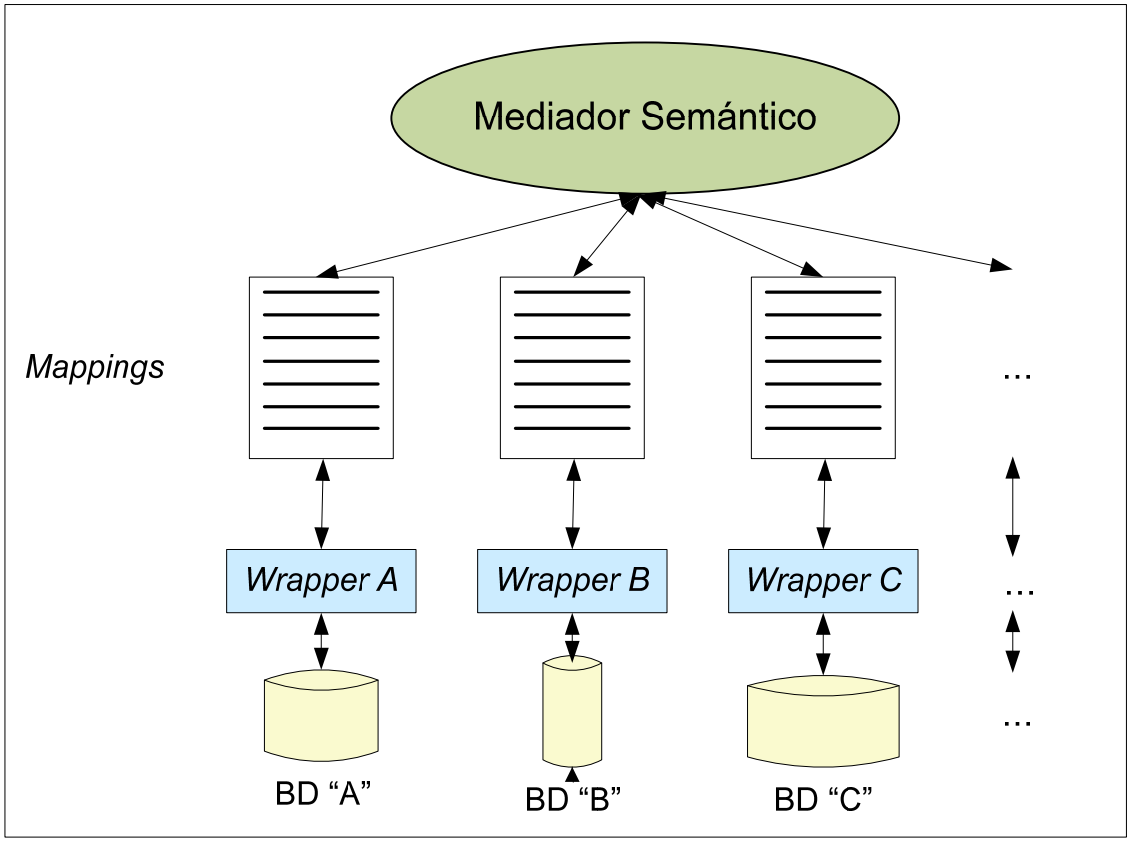

Fig 5-21 Arquitectura de los ejemplos

Cada una de estas Bases de Datos está cubierta por un *wrapper*. También existe un *mapping* por cada *wrapper* (ver figura Fig 5-21), de forma que los términos a nivel

conceptual formulados en una consulta se traducirán a términos en el nivel físico de cada fuente de datos.

El proceso de resolución de una consulta comprende varias fases:

- Se recibe la consulta en términos conceptuales.
- Se comprueba qué atributos y relaciones de la consulta están incluidos en cada uno de los *mappings* del sistema. Se divide la consulta en las diferentes subconsultas que atañen a cada *mapping*.
- Se traduce la subconsulta a términos físicos haciendo uso de las traducciones contenidas en los respectivos *mappings*.
- Se lanza la subconsulta a cada *wrapper* (siempre que exista subconsulta para ese *wrapper*) y se recogen los diferentes resultados de cada uno de ellos.
- Se integran los resultados:
    - Se vuelven a traducir los atributos, esta vez de términos físicos a conceptuales.
    - Se "reúnen" todos los resultados en un repositorio virtual que será destruido cuando se haya respondido a la consulta.
    - Se imponen las restricciones (si las hubiera) sobre ese repositorio que incluye todos los datos que tiene el sistema.
    - Se devuelven los resultados

El primero de los experimentos consiste en extraer un atributo de cada Base de Datos (ver figura Fig 5-22). Se realiza una consulta sobre edades y principios activos. Las edades están almacenadas en la Base de Datos "A" y los principios activos de los antibióticos en la Base de Datos "C". La Base de Datos "B" no contiene datos sobre edades o principios activos y por ello, al dividir la consulta, no se encuentra traducción en esa fuente.

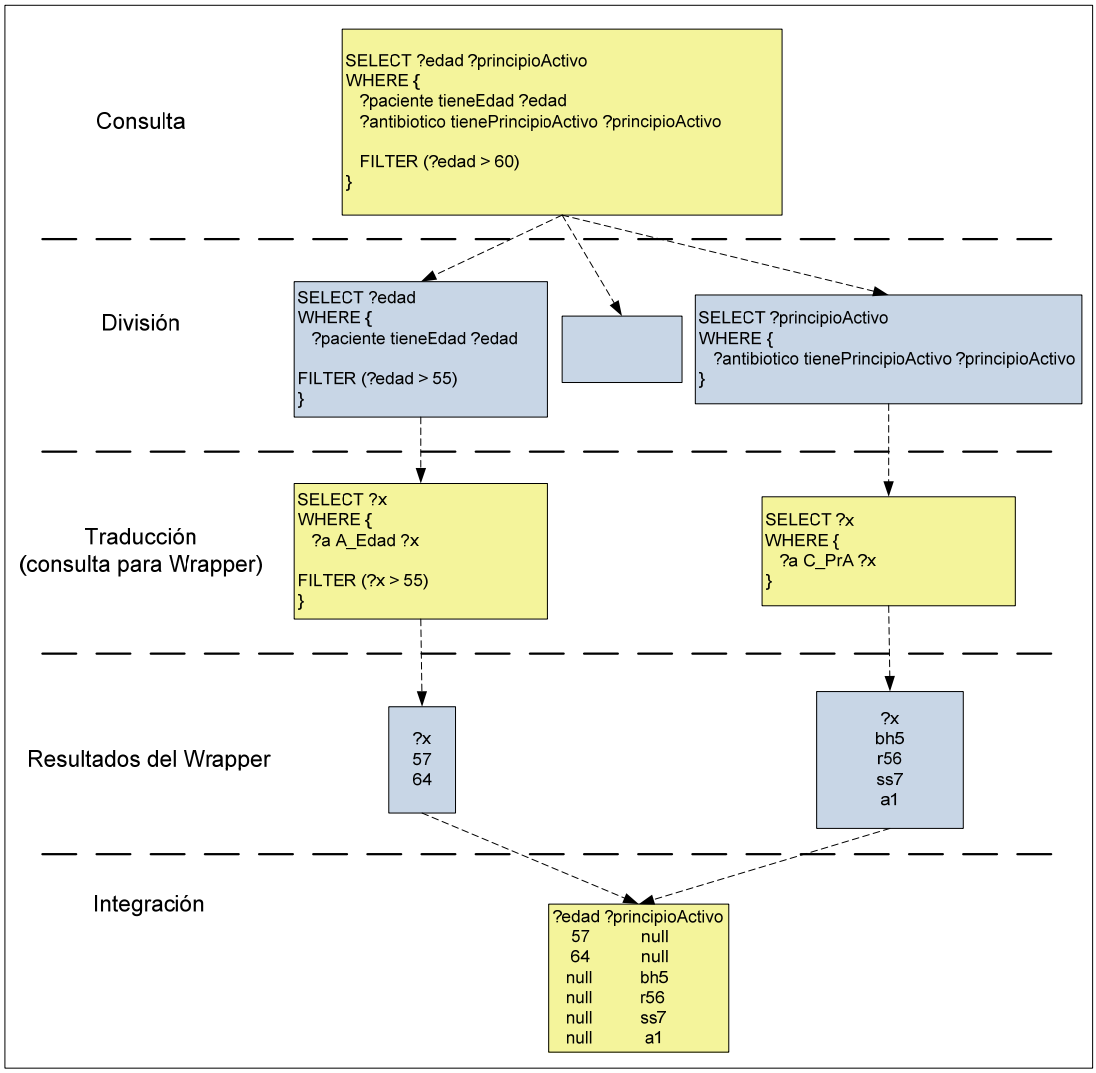

Fig 5-22 "SemanticMediator": existen varios *mappings* (I)

También se incluye la restricción de que la edad sea superior a 55 en la subconsulta para la Base de Datos "A" ya que sólo "A" tiene datos sobre la edad de un paciente. Se lanzan las subconsultas contra los respectivos wrappers que envuelven las Bases de Datos "A" y "C", se recogen los resultados y se integran. Al no tener atributos comunes, la integración consiste prácticamente en una unión de los distintos resultados (marcando a *null* los valores no encontrados).

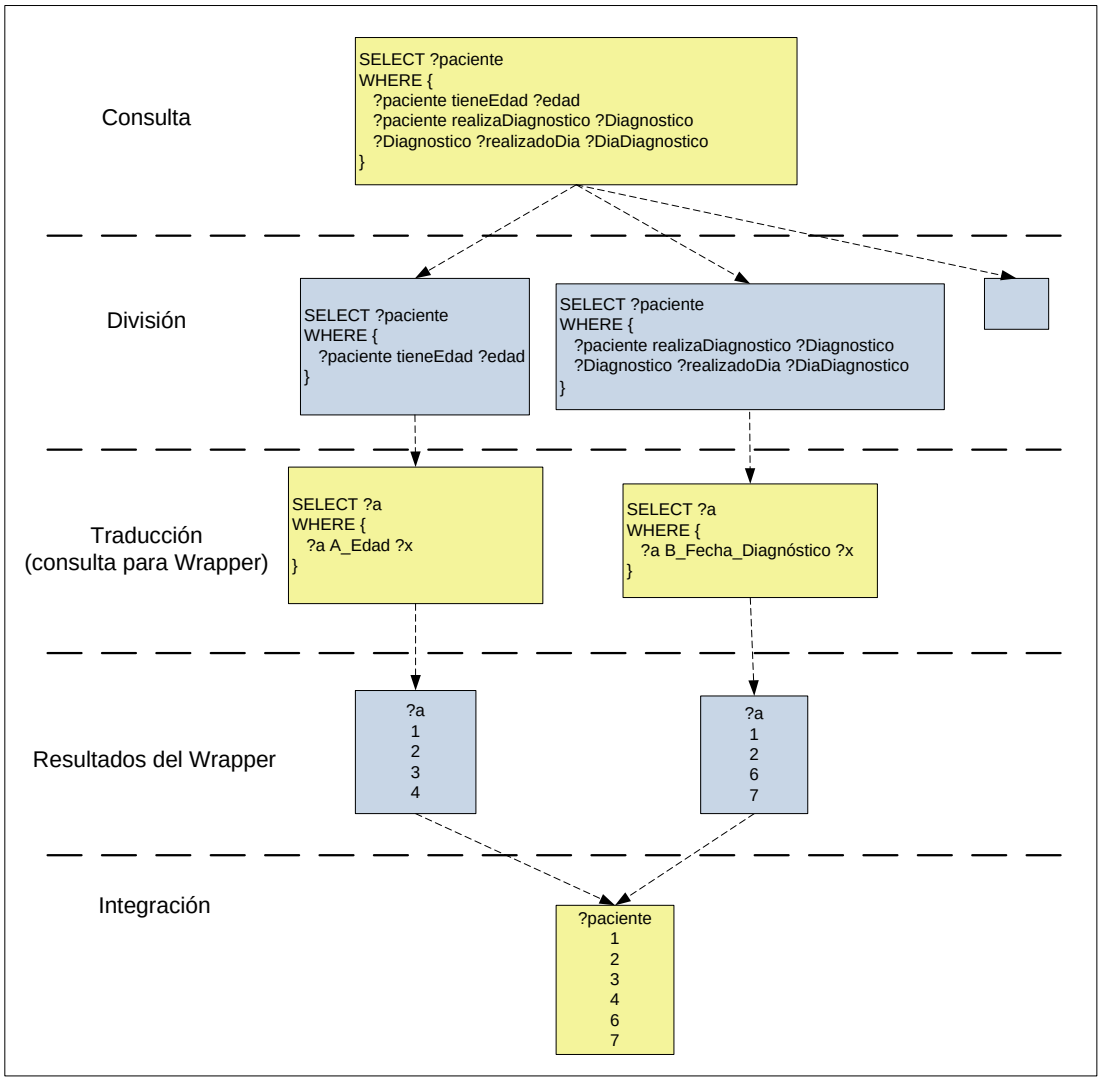

Fig 5-23 "SemanticMediator": existen varios *mappings* (II)

En el segundo experimento (ver figura Fig 5-23) con varios *mappings* se consulta por un atributo existente en varias Bases de Datos. Tanto en la Base de Datos "A" como en la "B" existen datos sobre pacientes (en "A" un paciente tiene edad y en "B" realiza un ingreso realizado en cierto día). La consulta se divide, se traducen a términos físicos cada una de las subconsultas, se lanzan contra los respectivos *wrappers* y se recogen los resultados. La diferencia con el caso anterior radica en que la integración no se limita a ser una unión de resultados, sino que se filtran los mismos para que no existan valores repetidos.

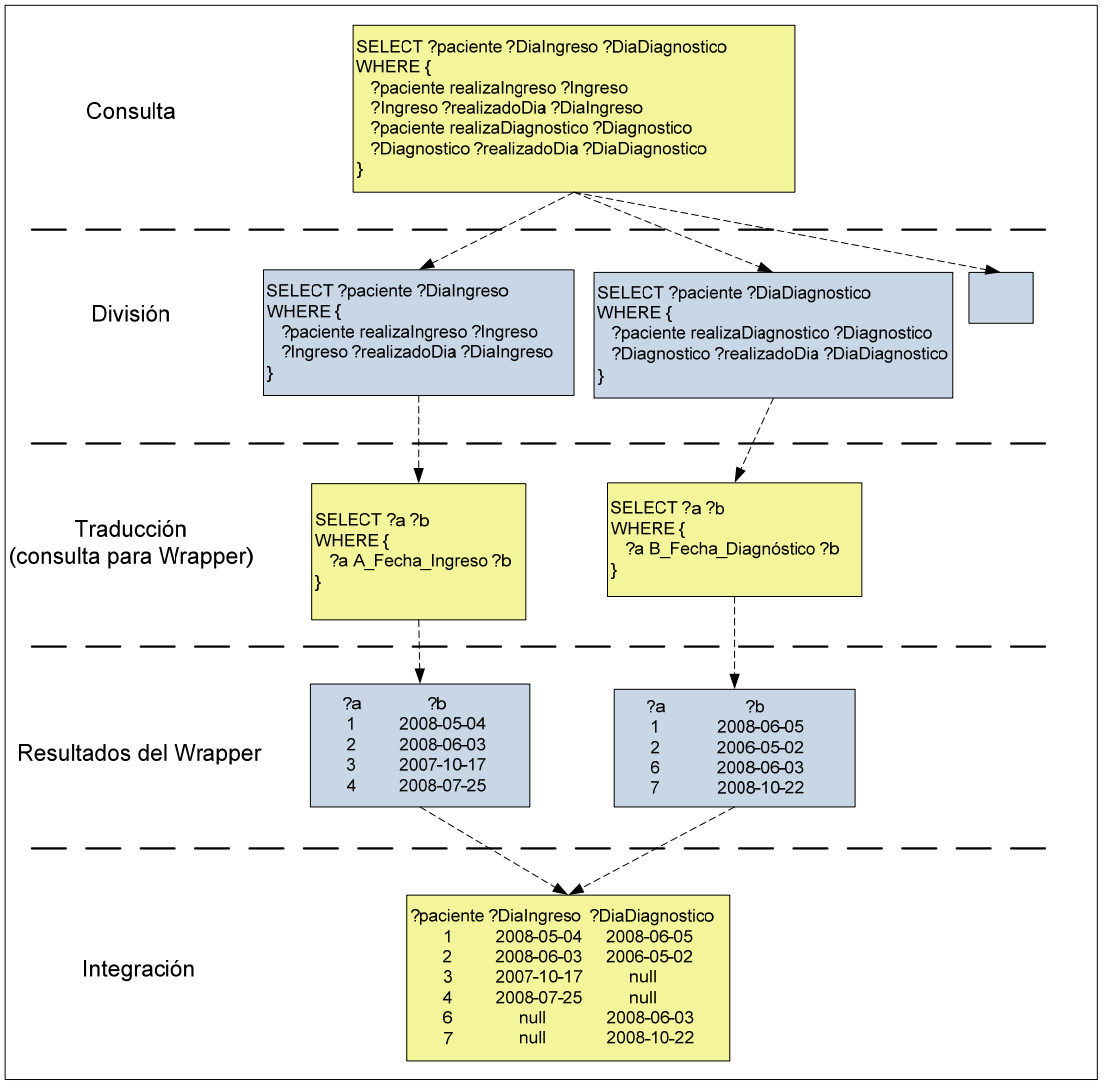

Fig 5-24 “SemanticMediator”: existen varios *mappings* (III)

En el tercer ejemplo se complica un poco más al preguntarse por tres atributos de los cuales uno se encuentra en las Bases de Datos “A” y “B”, otro sólo en “A” y otro sólo en “B”. Al dividir la consulta, como es lógico, dos atributos y sus respectivas relaciones se incluyen en las subconsultas respectivas. La traducción a términos físicos y recoger los resultados de los *wrappers* se realiza de la misma forma que en ejemplos anteriores. Por último, al realizar la integración ocurre que para el atributo ?paciente hay resultados provenientes de dos Bases de Datos y además tienen datos asociados como son ?DiaIngreso y ?DiaDiagnóstico respectivamente. Se realiza un filtrado de la información de forma correcta, asociando el conjunto de atributos por la variable ?paciente y marcando a *null* los valores no encontrados (ver figura Fig 5-24).

En el último experimento (ver figura Fig 5-25) se pregunta por los identificadores de los pacientes y su edad. Estos datos se recogen exclusivamente en la Base de Datos “A”, sin embargo la consulta establece una restricción y sólo se interesa por aquellos pacientes cuya fecha de ingreso sea posterior a la de diagnóstico, lo cual complica la consulta ya que la información sobre el día de ingreso se encuentra en la Base de Datos “A” y la del día de diagnóstico en “B”. Por este motivo en el proceso de dividir la consulta entre las distintas fuentes de datos, además de “repartir” en cada una los atributos y relaciones que le correspondan se añade a las variables de interés (las que se consultan) los días de ingreso y de diagnóstico. Nótese que la restricción original sobre que se haya realizado antes el diagnóstico que el ingreso no se ha “repartido” en ninguna subconsulta, ya que los dos atributos no pertenecen a la misma fuente de datos; esa restricción se guarda para una fase posterior en la que pueda ser aplicada. Una vez dividida la consulta, se traducen las subconsultas, se lanzan contra los wrappers y se recogen los resultados. Se “reúnen” todos esos resultados creando un repositorio para tal efecto y sobre él se impone la restricción que no se pudo imponer antes.

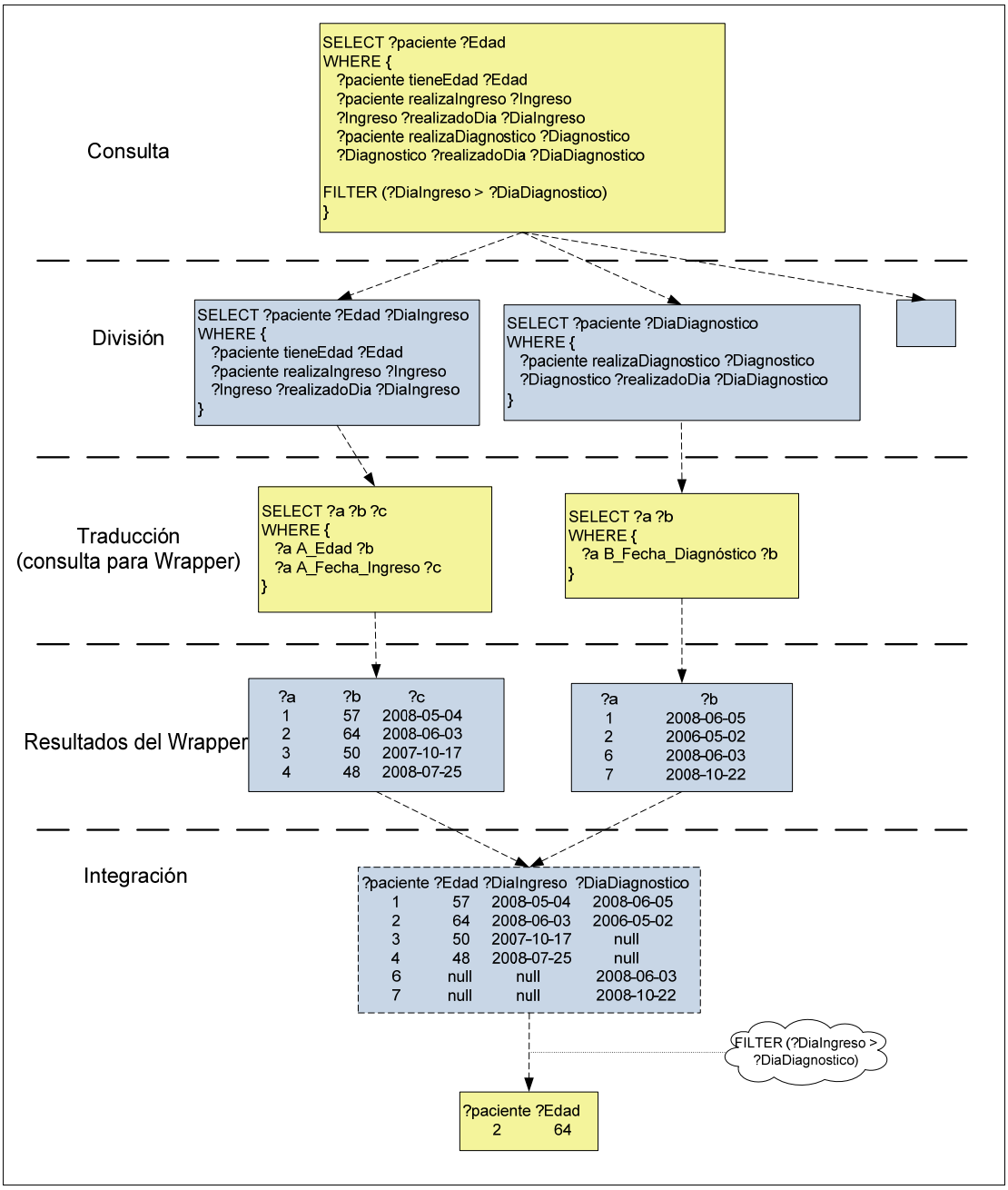

Fig 5-25 “SemanticMediator”: existen varios *mappings* (IV)

6 CONCLUSIONES Y LÍNEAS FUTURAS

6.1 CONCLUSIONES

El sistema presentado en este trabajo de fin de carrera tiene como objetivo ofrecer una capa de mediación semántica entre usuarios y fuentes de datos heterogéneas y distribuidas, dicha capa basada en tecnología Grid.

El enfoque propuesto, basado en la tecnología proporcionada por OGSA-DAI y Globus Toolkit ofrece una serie de servicios que dan unos resultados correctos a los objetivos planteados en la sección 1.2 del proyecto. En concreto se puede concluir que:

1. Se ha conseguido desarrollar una herramienta que gestione de forma transparente la respuesta a consultas sobre bases de datos heterogéneas y distribuidas. La respuesta a estas consultas se muestra en dos ficheros: uno con los resultados (en formato .csv) y otro con los metadatos asociados a los resultados.
2. Se permite la opción de que la herramienta envíe los dos ficheros con los resultados a un repositorio integrado en la plataforma.
3. Por último, el sistema cuenta con otra serie de servicios que dan respuesta a las necesidades de otras herramientas, como es el caso del servicio para la actualización de ficheros de correspondencias en el sistema para la *MappingTool*, o el servicio que lista los *wrappers* que hay disponibles para que la *QueryTool* efectúe las consultas.

En cuanto a las aplicaciones del sistema, cabe destacar que el proyecto es independiente del dominio, es decir, aunque actualmente se centra en el ámbito de la Biomedicina y la Bioinformática (dentro del proyecto ACGT), se puede trasladar a otras áreas. Simplemente bastaría generar una ontología sobre el dominio de interés concreto al que pertenecieran las fuentes de datos que se quisieran integrar.

A nivel personal, la realización de este trabajo de fin de carrera ha sido una experiencia muy positiva, tanto por los objetivos marcados los cuales eran novedosos para los conocimientos que adquirí durante la carrera; como por los problemas que se encontraron durante la realización del proyecto y las soluciones que se han aplicado.

6.2 LÍNEAS FUTURAS

A pesar de que la evaluación del sistema planteado en este trabajo de fin de carrera ha ofrecido resultados satisfactorios y prometedores, durante la realización del mismo han surgido varias ideas para efectuar diversas mejoras que podrían mejorar la calidad del sistema:

- Reconocer distintos lenguajes de representación de ontologías

  Actualmente el sistema es capaz de interactuar con ontologías en lenguaje OWL. Si existieran más módulos de reconocimiento de ontologías dentro del sistema o si el que ya existe reconociera más lenguajes de representación podrían utilizarse muchas más ontologías de acceso público existentes en la Web.

- Reconocer distintos lenguajes de consultas

  SPARQL (el lenguaje que soporta el sistema actualmente para la realización de consultas), es un lenguaje realmente útil para la recuperación de RDF/RDFS por permitir que las personas puedan centrarse en la información que quieren, sin tener en cuenta la tecnología de la base de datos o el formato utilizado para almacenar los mismos. Sin embargo no es útil si los datos no están almacenados en un formato RDF o no se dispone de algún *middleware* que traduzca a dicho formato, como ocurre en las bases de datos públicas.

- Mejorar el rendimiento del sistema:

  Computacionalmente hay ciertos aspectos que podrían mejorarse en la implementación del mediador semántico. Por ejemplo, el lanzamiento contra las distintas fuentes de datos de las subconsultas generadas por el sistema, podría realizarse de forma concurrente. También podría mejorarse el proceso de integración de las distintas fuentes de datos si se diera el caso, por ejemplo, de la existencia de “similitudes” entre los resultados de dos fuentes de datos. Por último, cabría estudiar la posibilidad de integrar los resultados a medida de que éstos fueran llegando al sistema en vez de esperar a que todos los *wrappers* hayan devuelto los resultados.

- Añadir nuevos métodos de integración:

  Sería interesante añadir nuevos métodos de integración en el sistema, que resultaran más interesantes si se usaran otras fuentes de datos distintas a las que se utilizan actualmente.

- Añadir seguridad al sistema:

  Como se ha comentado, todos los servicios del sistema se basan en el uso de la tecnología proporcionada por *GlobusToolkit*. Se planea elevar la versión que se usa actualmente, pues en la siguiente versión, se da soporte a la implantación de servicios seguros.

7 REFERENCIAS

[1]. Página principal del proyecto europeo Advancing Clinico-Genomic Trials on cancer ACGT. Disponible en: http://www.eu-acgt.org/. Accedido por última vez en diciembre de 2008.

[2]. Walter Sujansky, Heterogeneous database integration in biomedicine, Computers and Biomedical Research, v.34 n.4, p.285-298, August 2001

[3]. Ohno-Machado L, Boxwala AA, Ehresman J, Smith DN, Greenes RA.Avirtual repository approach to clinical and utilization studies: application in mammography as alternative to a national database. In: Proceedings American Medical Informatics Association Fall Meeting: Hanley & Belfus, 1997; 369–73.

[4]. Chen IA, Kosky A, Markowitz VM, Szeto E. OPM*QS: the object- protocol model multidatabase query system. Technical Report LBNL-38181, Lawrence Berkeley National Laboratory, 1995.

[5]. Sujansky W., Altman R. Toward a standard query model for sharing decision-support applications. Proceedings American Medical Informatics Association Fall Meeting 1994; 325–31.

[6]. R. Kimball. The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses, John Wiley. 1996.

[7]. A. P. Sheth e J. A. Larson. "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases", ACM Computing Surveys, 22(3): pp. 183-236. 1990

[8]. David Pérez del Rey. Un modelo de integración y preprocesamiento de información distribuida basado en ontologías. Tesis Doctoral. Disponible en: http://oa.upm.es/1052/01/DAVID_PEREZ_DEL_REY.pdf. Accedido por última vez en diciembre de 2008.

[9]. Ritter, O., Kocab, P., Senger, M., Wolf, D., y Suhai, S. (1994). “Prototype implementation of the integrated genomic database”. Computers and Biomedical Research, vol. 27, nr. 2, pp. 97-115

[10]. Etzold, T., y Argos, P. (1993). “SRS, and Indexing and Retrieval Tools for Flat Files Data Libraries”. Computer applications in the biosciences vol.9: pp. 49-

57. Disponible en: http://srs.ebi.ac.uk. Accedido por última vez en diciembre de 2008.

[11]. Wiederhold, G. (1992). Mediators in the Architecture of Future Information Systems. IEEE Computer, 25(3):38-49.

[12]. Miguel García Remesal. Método de adquisición de modelos de dominio a partir de corpus textuales y su aplicación en la integración de bases de datos y fuentes de información. Tesis doctoral.

[13]. Silvescu, A., Reinoso-Castillo, J., Honavar, V. (2001). Ontology-Driven information extraction and knowledge acquisition from heterogeneous, distributed, autonomous data sources. En Proceedings of the IJCAI 2001.

[14]. Sheth and Larson (1990). "Federated Database Systems for Managing Distributed, Heterogenous, and Autonomous Databases". ACM Computing Surveys Vol 22, No.3: 183-236.

[15]. Ruggia, R. Incidencia de la Calidad y Semántica de Datos en Sistemas de Información Federados. Disponible en: http://www.fing.edu.uy/inco/grupos/csi/esp/Proyectos/Sico/CLEI2002_FDB&Q.pdf. Accedido por última vez en diciembre de 2008.

[16]. Miled, Z. B., Li, N. y Bukhres, O. (2005). “BACIIS: Biological and Chemical Information Integration System”. Journal of Database Management, 16(3):72-85.

[17]. Hamosh, A., Scott, A.F., Amberger, J., Valle, D., y McKusick, V.A. (2000). “Online Mendelian Inheritance in Man (OMIM)”. Hum Mutat 15(1):57-61.

[18]. Berman, H.M., Westbrook, J., Feng, Z., Gilliand, G., Bhat, T.N., Weissig, H., Shindyalov, I.N. y Bourne, P.E. (2000). “The Protein Data Bank”. Nucleic Acids Res. 28, pp. 235-242.

[19]. Falquet, L., Pagni, M., Bucher, P., Hulo, N., Sigrist, C.J., Hofmann, K., y Bairoch, A. (2002). “The PROSITE database, its status in 2002”. Nucleic Acids Res. 30 pp. 235-238.

[20]. Baker, P.G., Brass, A., Bechhofer, S., Goble, C., Paton, N., y Stevens, R. (1998). “TAMBIS – Trasparent Access to Multiple Bioinformatics Information Sources”. Intelligent Systems for Molecular Biology, 6:25-34.

[21]. Mena, E., Kashyap, V., Sheth, A. y Illarramendi, A. (1996). “OBSERVER: An Approach for Query Processing in Global Information Systems base don

Interoperation across Pre-existing Ontologies. Conference on Cooperative Information Systems, pp. 14-25.

[22]. Visser, U., Vöguele, T., y Schliede, C. (2002). "Spatio-Terminological Information Retrieval using the BUSTER system". Vol.1 of Environmental Communication in the Information society, pp. 93-100.

[23]. Pérez-Rey, D., Maojo, V., García-Remesal, M., Alonso-Calvo, R., Billhardt, H., Martín-Sánchez, F., Sousa, A. (2006): ONTOFUSION: Ontology-based Integration of Genomic and Clinical Databases. Computers in Biology and Medicine 36(7-8):712-30.

[24]. JAVA (Sun microsystems): [Sun03] *Java 2 SDK, Standard Edition Documentation, version 1.4.2*. Sun Microsystems, 2003.

[25]. Características del lenguaje JAVA. Disponible en: http://www.iec.csic.es/criptonomicon/java/quesjava.html. Accedido por última vez en diciembre de 2008.

[26]. Extensible Markup Language XML. Disponible en: http://www.w3.org/TR/REC-xml/. Accedido por última vez en diciembre de 2008.

[27]. XML. Disponible en: http://trevinca.ei.uvigo.es/~txapi/espanol/proyecto/superior/memoria/node153.html. Accedido por última vez en diciembre de 2008.

[28]. Resource Description Framework (RDF). Disponible en: http://www.w3.org/RDF/. Accedido por última vez en diciembre de 2008.

[29]. Lenguaje de recuperación SPARQL. Disponible en: http://geocities.com/recuperacioninformacionorganiza/sparql.html. Accedido por última vez en diciembre de 2008.

[30]. SPARQL Query Language for RDF. Disponible en: http://www.w3.org/TR/rdf-sparql-query/. Accedido por última vez en diciembre de 2008.

[31]. Fundamentos de la Web Semántica: RDF avanzado. Pablo R. Fillotrani. Disponible en: http://cs.uns.edu.ar/~prf/teaching/FSW07/clase6.pdf. Accedido por última vez en diciembre de 2008.

[32]. The D2RQ Platform v0.5.1 - Treating Non-RDF Relational Databases as Virtual RDF Graphs. Disponible en: http://www4.wiwiss.fu-berlin.de/bizer/D2RQ/spec/. Accedido por última vez en diciembre de 2008.

[33]. OWL. Disponible en: http://es.wikipedia.org/wiki/OWL. Accedido por última vez en diciembre de 2008.

[34]. OWL Web Ontology Language. Disponible en: http://www.w3.org/TR/2004/REC-owl-features-20040210/#s1.2. Accedido por última vez en diciembre de 2008.

[35]. Base de Datos. Disponible en: http://es.wikipedia.org/wiki/Base_de_datos. Accedido por última vez en diciembre de 2008.

[36]. Bases de Datos. Disponible en: http://www.monografias.com/trabajos11/basda/basda.shtml. Accedido por última vez en diciembre de 2008.

[37]. Ventajas e inconvenientes de los sistemas de Bases de Datos. Disponible en: http://www3.uji.es/~mmarques/f47/apun/node7.html. Accedido por última vez en diciembre de 2008.

[38]. Quiroz J. “El modelo relacional de bases de datos”. Boletín de Política Informática Num. 6. 2003. Disponible en: http://www.inegi.gob.mx/inegi/contenidos/espanol/prensa/Contenidos/Articulos/tecnologia/relacional.pdf. Accedido por última vez en diciembre de 2008.

[39]. Groff, James R; Weinberg, Paul N. “SQL: The Complete Reference”, Second Edition. OSBORNE/MCGRAW. 09/2002.

[40]. SQL. Disponible en: http://es.wikipedia.org/wiki/Celda_activa. Accedido por última vez en diciembre de 2008.

[41]. Guía breve de Servicios Web. Disponible en: http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb. Accedido por última vez en diciembre de 2008.

[42]. Trabajo Fin de Carrera: "Sistema Global de Integración y Preprocesamiento de Bases de Datos Heterogéneas basado en Ontologías", Elena Aguilera, Facultad de Informática, UPM.

[43]. Servicio Web. Disponible en: http://es.wikipedia.org/wiki/Servicio_Web. Accedido por última vez en diciembre de 2008.

[44]. Fran Berman, Anthony J.G. Hey, Geoffrey Fox (2003). Grid Computing: Making The Global Infrastructure a Reality. Wiley. Disponible en: http://www.grid2002.org/. Accedido por última vez en diciembre de 2008.

[45]. Introducción a la Computación Grid. Borja SotoMayor. Disponible en: http://people.cs.uchicago.edu/~borja/lectures/IntroduccionGrid.pdf. Accedido por última vez en diciembre de 2008.

[46]. Ian Foster, Carl Kesselman (1999). La Malla: libro azul para una nueva infraestructura informática (The Grid: Blueprint for a New Computing Infrastructure). Morgan Kaufmann Publishers. ISBN: 1558604758.

[47]. Defining the Grid: Open Grid Services Architecture. Disponible en: www.ggf.org/GGF17/materials/316/OGSA%20keynote%2020060510.ppt. Accedido por última vez en diciembre de 2008.

[48]. Globus Toolkit. José Arturo García Monroy. E.T.S.I.Telecomunicación – UPM. Disponible en: http://internetng.dit.upm.es/joe/Art/Globus.pdf. Accedido por última vez en diciembre de 2008.

[49]. Open Grid Services Architecture WG (OGSA-WG). Disponible en: https://forge.gridforum.org/projects/ogsa-wg. Accedido por última vez en diciembre de 2008.

[50]. Web Service Resource Framework. Disponible en: http://es.wikipedia.org/wiki/Web_Service_Resource_Framework. Accedido por última vez en diciembre de 2008.

[51]. Página principal de “the Globus Alliance”. Disponible en: http://www.globus.org/. Accedido por última vez en diciembre de 2008.

[52]. The Globus Toolkit 4. Disponible en: http://gdp.globus.org/gt4-tutorial/multiplehtml/ch01s04.html. Accedido por última vez en diciembre de 2008.

[53]. Ian Foster, “What is the Grid? A three point check list”, Grid Today, 2002

[54]. Página Web de OGSA-DAI. Disponible en: http://www.ogsadai.org.uk/ . Accedido por última vez en diciembre de 2008.

[55]. Documentación de la arquitectura de OGSA-DAI. Disponible en: http://www.ogsadai.org.uk/documentation/ogsadai-wsrf-2.2/doc/background/architecture.html. Accedido por última vez en diciembre de 2008.

[56]. Ontología. Disponible en: http://es.wikipedia.org/wiki/Ontolog%C3%ADa Accedido por última vez en diciembre de 2008.

[57]. Barchini G., Álvarez M., Herrera S. “Sistemas de Información: Nuevos escenarios basados en Ontologías.” Journal of Information Systems and Technology Management. Vol. 3, No. 1, 2006, p.3-18. Disponible en: http://www.jistem.fea.usp.br/index.php/jistem/article/viewFile/26/42. Accedido por última vez en diciembre de 2008.

[58]. Gomez-Perez A., Corcho O., Fernandez-Lopez M. “Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web”. Springer; Primera edición 2004.

[59]. IEEE Recommended Practice for Software Requirements Specification. IEEE Standard, 1998

[60]. Larman, C. (2002). “UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado. Segunda Edición”. Prentice Hall, 2002

[61]. Martín, Luis; Anguita, Alberto; Jiménez, Ana; Crespo, José, "Enabling Cross Constraint Satisfaction in RDF-Based Heterogeneous Database Integration," *Tools with Artificial Intelligence, 2008. ICTAI '08. 20th IEEE International Conference on* , vol.2, no., pp.341-348, 3-5 Nov. 2008

APÉNDICE I: MANUAL DE INSTALACIÓN DEL MEDIADOR SEMÁNTICO

En esta sección se explicarán los pasos a seguir para instalar el Mediador Semántico y comprobar la correcta instalación.

El Mediador semántico es dependiente de una serie de paquetes software que necesitan ser instalados previamente. Estos paquetes, y las respectivas versiones utilizadas para este Trabajo de Fin de Carrera, se comentarán a continuación.

Apache Ant

Apache Ant es una herramienta utilizada en programación para la realización de tareas mecánicas y repetitivas. Es similar a la herramienta "Make", pero sin las engorrosas dependencias con el sistema operativo. Se basa en archivos de configuración XML y está construida en Java.

La versión utilizada en este trabajo ha sido la 1.7.0 descargable desde la página oficial de Apache Ant (http://ant.apache.org).

La instalación de Apache Ant es muy sencilla. Si se descarga la distribución binaria, bastará con descomprimirla en el directorio C:\ y con añadir la variable de entorno C:\apache-ant-1.7.0\bin.

Se comprueba la instalación de Apache Ant tal y como se muestra en la figura Fig A-0-1.

Fig A-0-1 Correcta instalación de Apache Ant

Instalación de *Globus Toolkit*

Globus Toolkit es un software libre desarrollado por la *Globus Alliance* y que se utiliza para construir Grids computacionales.

La versión utilizada en este trabajo ha sido la 4.0.3 en su versión para servicios Web (ws-core.4.0.3).

Se puede descargar dicha versión desde la página oficial de Globus Toolkit. La usada en la realización de este trabajo ha sido la distribución binaria. Y los pasos a seguir para la instalación han sido:

1.- Descomprimir en C:\.

2.- Añadir la variable de entorno GLOBUS_LOCATION

```
set GLOBUS_LOCATION=C:\ws-core-4.0.3
```

3.- Comprobar que se puede "levantar" el contenedor de *Globus Toolkit* donde se alojarán los servicios del sistema. Para ello, ejecutar:

```
globus-start-container –nosec –debug
```

La salida a esta ejecución debería ser la de la figura Fig A-0-2.

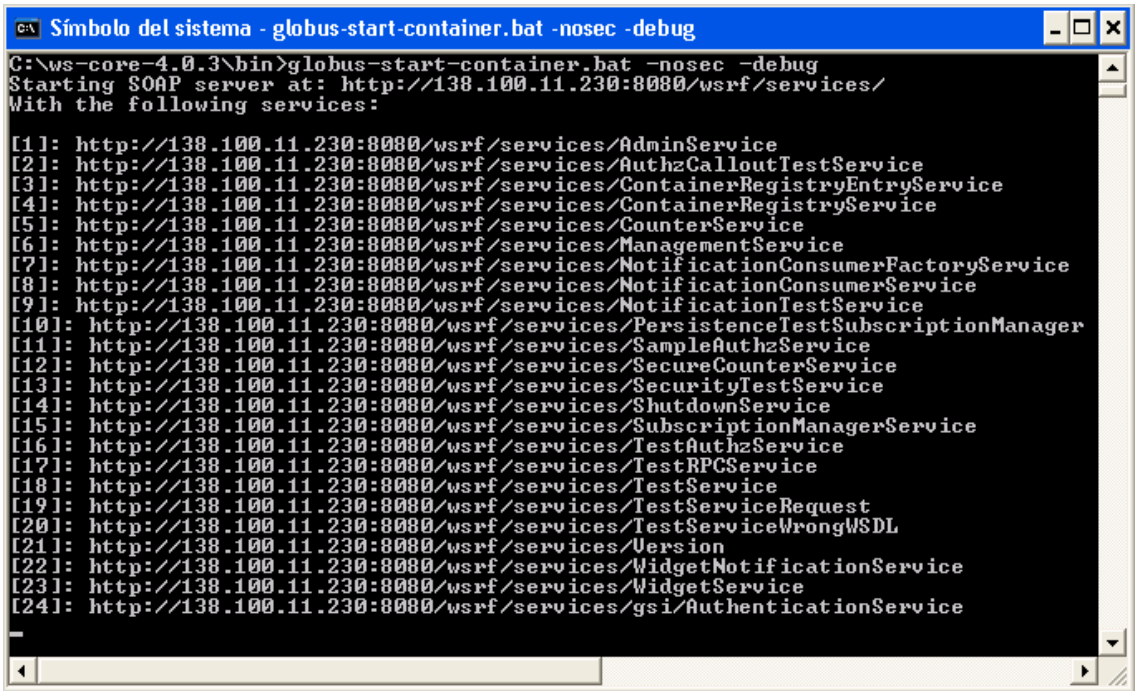

Fig A-0-2 Ejecución del contenedor de *Globus*

Se puede encontrar más información sobre *Globus Toolkit* en la sección 2.2.8.4.

Instalación de OGSA-DAI

OGSA-DAI es un middleware para asistir en el acceso e integración de datos desde distintas fuentes de datos a través del Grid. Más información sobre este software se puede encontrar en la sección 2.2.8.5.

La versión de OGSA-DAI utilizada durante la realización del proyecto ha sido la 2.2, la cual es compatible con la versión utilizada de *Globus Toolkit*.

Los pasos seguidos para la instalación correcta de OGSA-DAI son:

1.- Descomprimir en C:\ la distribución binaria del software.

2.- Utilizar el entorno gráfico para la instalación de OGSA-DAI (figuras Fig A-0-3 y Fig A-0-4)

```
C:\ogsadai-wsrf\ant guiInstall
```

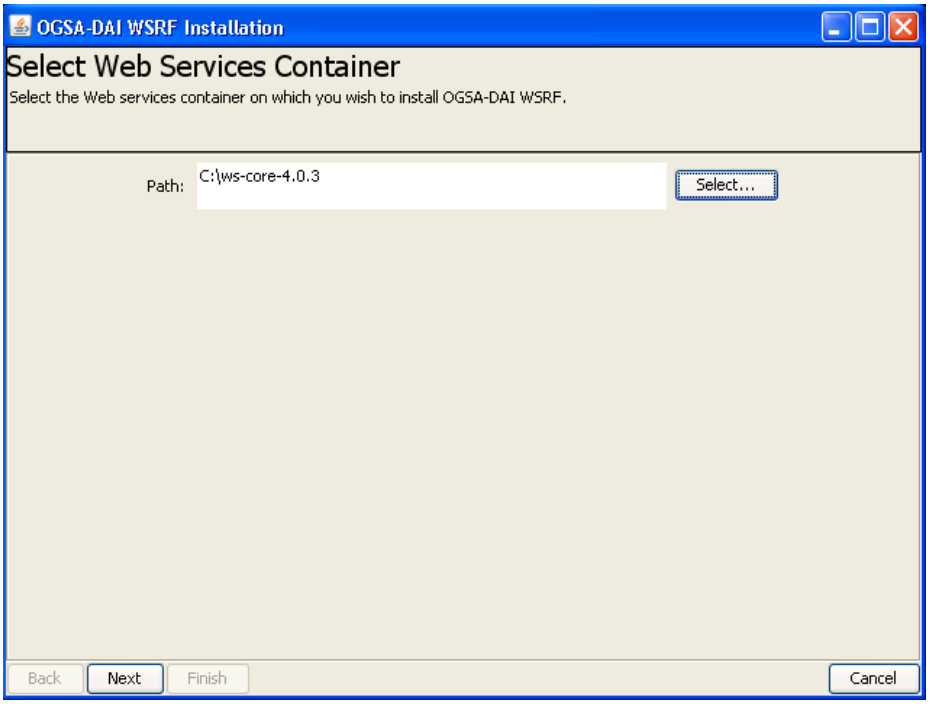

Fig A-0-3 Instalación de OGSA-DAI (I)

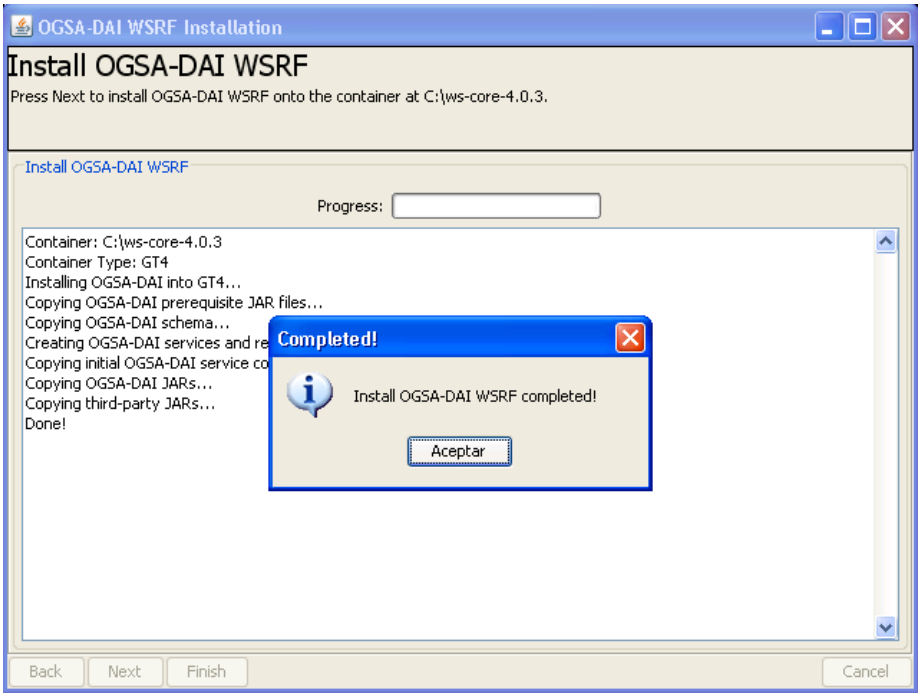

Fig A-0-4 Instalación de OGSA-DAI (II)

3.- Instalar un nuevo *Data Service* utilizando el entorno gráfico (figura Fig A-0-5)

```
C:\ogsadai-wsrf\ant guiDeployService
```

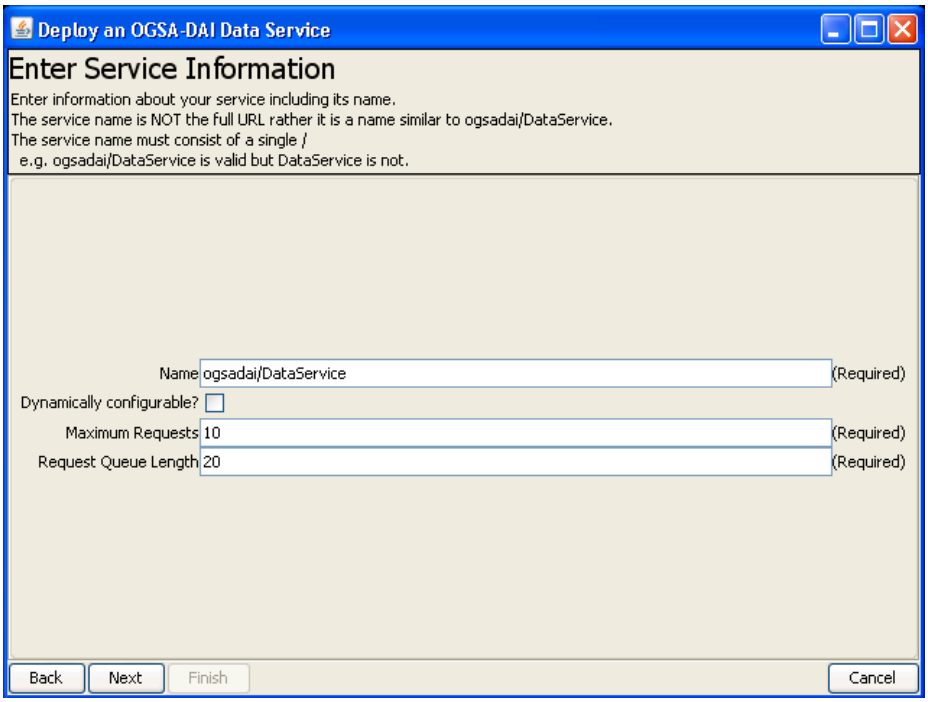

Fig A-0-5 Instalación de OGSA-DAI (III)

4.- Reiniciar el contenedor de *Globus Toolkit* y comprobar que todo ha ido correctamente (ver figura Fig A-0-6).

```
C:\ogsadai-wsrf\ant listResourcesClient
 -Ddai.url=http://localhost: 8080/wsrf/services/ogsadai/DataService
```

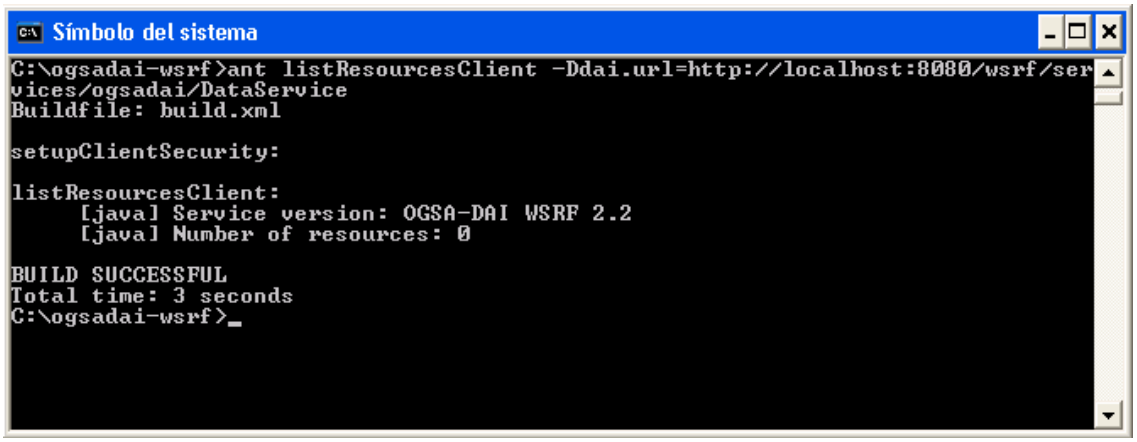

Fig A-0-6 Instalación de OGSA-DAI (IV)

5.- Modificar el fichero:

```
C:\ws-core-4.0.3\etc\ogsadai_wsrf\_ogsadai_DataService.dsr.xml
```

Para incluir dos nuevos recursos en el *Data Service* (ver Figura Fig A-0-7):

- loadFiles: es el recurso responsabilizado de actualizar los ficheros en el repositorio de DMS.
- SemanticMediator: es el recurso que se encarga de realizar la tarea de mediación semántica en sí.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- (c) International Business Machines Corporation 2005. -->
<!-- (c) University of Edinburgh, 20025. -->
<!-- See OGSA-DAI-Licence.txt for licencing information. -->
- <dataServiceResources xmlns="http://ogsadai.org.uk/namespaces/2005/10/config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <dataServiceResource name="loadFiles" />
  <dataServiceResource name="SemanticMediator" />
</dataServiceResources>
```

Fig A-0-7 Instalación de OGSA-DAI (V)

6.- Colocar los ficheros del esquema de los recursos en la carpeta correspondiente:

```
C:\ws-core-4.0.3\share\schema\ogsadai\xsd\activities\loadFilesSchema.xsd
```

```
C:\ws-core-4.0.3\share\schema\ogsadai\xsd\activities\SemanticMediatorWIPSchema.xsd
```

También los esquemas de la actividad para actualizar los ficheros en el repositorio DMS:

C:\ws-core-4.0.3\share\schema\ogsadai\xsd\activities\deliver_files_to_dms.xsd
C:\ws-core-4.0.3\share\schema\ogsadai\xsd\activities\deliver_to_dms.xsd
C:\ws-core-4.0.3\share\schema\ogsadai\xsd\activities\dms_file_info_to_xml.xsd

7.- Colocar las carpetas de configuración en las actividades en la ruta correspondiente.

C:\ws-core-4.0.3\etc\ogsadai_wsrf\loadFiles
C:\ws-core-4.0.3\etc\ogsadai_wsrf\SemanticMediator

8.- Descomprimir las librerías necesarias en la carpeta \lib de globus (C:\ws-core-4.0.3\lib en este caso).

9.- Copiar la carpeta ".globus" con los certificados de autenticación en el directorio HOME (Figura Fig A-0-8).

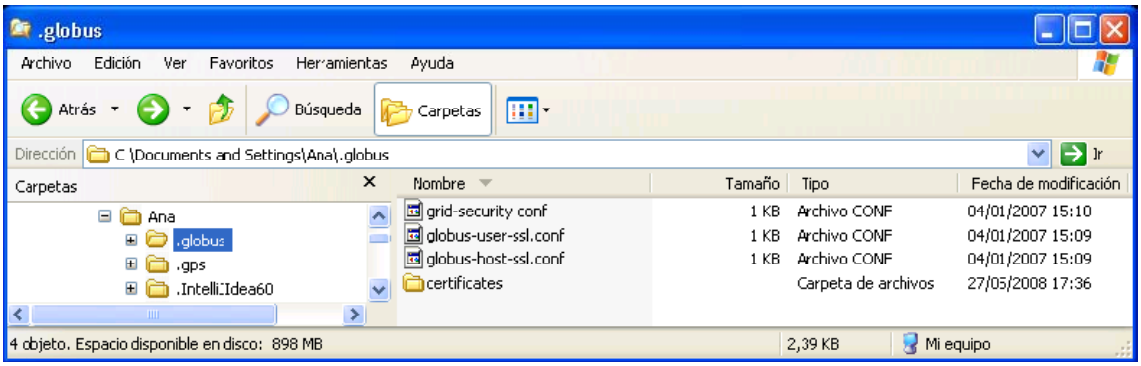

Fig A-0-8 Instalación de OGSA-DAI (VI)

10.- Por último, reiniciar el contenedor de globus y comprobar la ejecución correcta de los servicios (ver Figura Fig A-0-9).

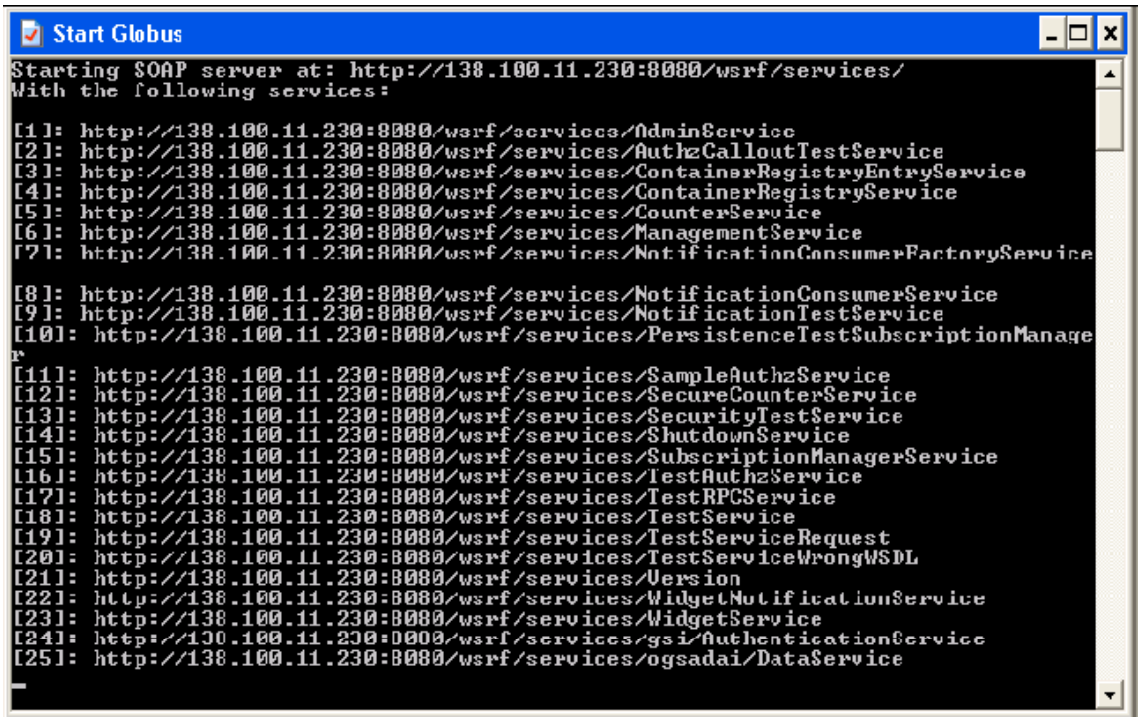

Fig A-0-9 Instalación de OGSA-DAI (VII)

APÉNDICE II: ARTÍCULO PUBLICADO

En las siguientes páginas se muestra un artículo [61] publicado en noviembre del 2008 en el Congreso “Tools with Artificial Intelligence, 2008, ICTAI ´08, 20th IEEE International Conference on”.

Enabling Cross Constraint Satisfaction in RDF-based Heterogeneous Database Integration

Luis Martín, Alberto Anguita, Ana Jiménez, José Crespo

Abstract— **The problem of database integration has been widely tackled through different approaches. While data transformation based systems, such as Data Warehouses, reached the acceptation of the industry during the 80's, in the last decade query translation based approaches have gained popularity given their adequacy to dynamic domains. While the former are based on gathering actual data in central repositories, the latter allow data to remain in the original databases. There still exist several issues to be tackled in query translation, mainly if no ad-hoc schema is described, such as problems with scalability and query processing. In this paper we describe a complete database integration and semantic mediation approach, borrowing techniques from both data transformation and local as view data translation methods, and addressing the cross referencing problem. This work has been carried out in the framework of ACGT (Advancing Clinico-Genomic Trials on Cancer) project, supported by the European Commission.**

I. Introduction

Dynamic data environments, such as those related with many research activities, require database integration approaches that leverage the complexity in terms of data scalability and performance, as well as a means to maintain access to up to date information in the databases. For this reason, query translation based approaches have gained popularity in the last decade, even when some very important problems have not been already solved. There exist two main different ways to tackle query translation, namely Global as View[1] and Local as View[2]. In Global as View, an ad-hoc schema is created representing the complete integrated database set. By contrast, in Local as View standalone descriptions of the sources are built by means of a global model. When dealing with unstable database integration environments—i.e. where database schemas can change, or new databases can arrive into the system— Local as View based approaches behave better, since there is no need of updating the global schema. However, this type of approach leads to several problems that need to be taken into consideration. The main issue is scalability: in the worst case, all the local views need to be inspected to find the translation of each element in the original query. Another problem is related to expressiveness: the global model is a description of the domain, not and ad-hoc description of the integrated repository (as in Global as View). This can lead to conflicts when a user formulates a query and what is queried is not contained in the underlying databases. Traditionally, Local as View approaches describe the way to translate the query, leaving aside the actual integration of results. In previous works, the data integration process has been defined as the union of the result sets retrieved from the produced queries. This implies that the simple union of the tables is equivalent to the results expected in the original query. Given the fact that most common query translation algorithms produce queries containing the variables of interest of the original query, this approach should work properly. However, there exist two situations where this is not so simple: a) there are variables in the original query translated to more than one variable in more than one database, 2) the original query contains constraints implying values from different databases (cross-reference, aggregation…). It is not possible to satisfy such kind of constraints by simply using a union operation on the retrieved results. Some systems partially cover these cases by supporting the *join* operation of semantically equivalent fields. In our system, we address these issues by creating a projection of the intermediate results retrieved from the original databases, comprising an RDF repository. Final results are retrieved using a query produced during the query translation process together with the dedicated queries for the underlying depositories. The results of this query are equivalent to the data expected to be retrieved from the original one.

Our approach is based on the utilization of an ontology, acting as global model. The global schema is extracted from this ontology, using its underlying RDF Schema. This global schema has two main roles in the mediation process: 1) acting as semantic framework for the mapping process, and 2) describing the complete universe of queries for the mediated database set. The use of ontologies for database integration has been widely studied and implemented in projects such as Ontofusion—carried out in our research lab [3] —, TSIMMIS [4] and KAON [5] among others.

The main issue to tackle when trying to implement tools based on a Local as View approach is performance. It is known that query translation in the latter has NP-Hard complexity. This leads to a scalability problem difficult to cope with. In our tool, we restrict the set of queries by creating user profiles. These profiles are based in previously gathered user requirements, which are used during the mapping process to create customized integrated database sets. Given that the mapping process in Local as View is the less complex one among all query translation approach, updates are feasible if profiles need to be changed. Although

Manuscript received June 30, 2008. This work was supported in part by the Ministry of Education, Spain, under Grants TSI2006-13021-C02-01 and TIN2007-61768, and the European Commission, under the ACGT project.

All authors are with the Polytechnic University of Madrid, Campus de Montegancedo, Boadilla del Monte, CP 28660 Spain (corresponding author contact information: Luis Martín, phone: 0034-91336-7467; fax: 0034-91336-7467; e-mail: lmartin@ informed.dia.fi.upm.es).

not all the queries are available, the profile should allow a user to formulate all the different queries he/she may need in his/her work. Another problem that users frequently find when dealing with a Local as View based mediation system is the complexity of the global schema. In the ideal case, this schema is nothing but a description of the domain of data—e.g. an ontology or the combination of several ones—. The schema exposed by our tool is indeed restricted to the allowed queries, being much simpler to understand and navigate.

This paper is organized as follows. Section 2 presents a state of the art review in the field. Section 3 introduces the methods used in this system. Section 4 describes the tools developed. Section 5 shows a set of experiments illustrating the systems behavior. Finally, section 6 points out our conclusions.

II. Background

There exists a plethora of papers dedicated to the subject of heterogeneous database integration. Most systems designed for this purpose follow a mediation-based approach, where a global schema is used to define the space of possible queries that the system is able to cope with. In addition, mappings between the global schema and the schemas of the integrated sources are stored. These mappings contain the necessary information to translate a query in terms of the global schema—i.e. global query—to a set of queries in terms of local schemas. Global as View and Local as View are the two existing approaches to define the mappings. The final result of a global query is obtained by merging the results of the generated queries. Most papers in this area cover the problem of producing queries for the physical databases, but make little to none mention about how actual data are merged to produce a unique result set.

Ullman [6] reviews the theoretical concepts that surround the query translation process in view-based integration systems. He considers global queries as a conjunction of predicates—no mention of additional constrains that bound the solutions to such predicates is made. The process of answering a query is described as a search of combinations of views contained in the original conjunction of predicates. Each of these combinations provides a partial solution to the original query. The procedure to obtain the final solution is stated in the paper as performing "the union of all these partial solutions", but no further details on this topic are given. Xu [7] describes his own approach for a data integration system using a mixture of LaV and GaV approaches for query reformulation, however no mention can be found about how the final result is computed. Saw [8] presents his work for integrating heterogeneous XML data sources, focusing on coping with the structural differences of the schemas to integrate. No mention is made about how final results are produced.

Some works briefly mention the process of obtaining a result to the original query by performing either a union of partial results—at most, a join operation is proposed. Halevi [9] proposes his bucket algorithm for performing query decomposition, stating that the result of this process is *"the union of two conjunctive queries"*. Pottinger performs a similar statement when describing her Minicon algorithm for query translation [10]. The same idea can be found in [11], where Cali describes his system for performing semantic integration of heterogeneous sources. Xiao, in his description of his own approach for integration of heterogeneous XML sources [12], states that the partial results are *"integrated (by using union) to produce the answer to q"*. Lehti, in his paper about integration of XML sources using OWL as global schema [13], mentions the necessity of defining join conditions when merging data from different sources. Camillo [14] focuses on the translation mechanism for producing a set of queries from a global query, mentioning also the necessity of including a mechanism *"to unify query results coming from several XML sources into a single query result in accordance to the global schema"*.

Other systems do take into account global integrity constraints during the process of query processing and result merging—to cope with the problem of inconsistencies between sources. Proper join operations are performed when partial results share semantically equivalent fields. Lenzerini [15] mentions the inconsistent sources issue in integration environments, but makes no mention of cross-constrains solving. In [16], a method for merging partial results using integrity constrains is presented, however only cases where all integrated sources share the same schemas are described. In [17], a method for performing query processing under primary and foreign keys restrictions over the global schema is described. Amann et al. [18] present an approach for integrating heterogeneous XML sources. In it, global and local schemas are populated with key values. This allows deciding, when possible, whether instances from different sources refer to the same entity, and thus require a join operation when merging the results. Jian et al. [19] use union and join operations to generate the final result from a set of partial results obtained from accessing the underlying databases.

Some systems generate query processing plans which include relational algebra operators to merge the data obtained from local sources. In DISCO [20], scan, project and join operators over partial results are used in order to compose the final answers to user queries. Mena et al. [21] propose a system which performs data integration by storing partial results in an auxiliary SQL database, which is later queried using extended relational algebra operators in order to perform proper union of data. This system is however limited to relational databases which significantly limit its scope.

Nevertheless, none of the mentioned approaches feature full cross-reference constrains in queries over the global schema. They consider semantic equivalence of fields only when these represent keys in the global schema, in order to avoid inconsistencies between sources in the final results.

III. Methods

Database integration can be divided in a set of different sub problems, tackled by different approaches. Among the most important sub problems we have identified schema

level heterogeneity, instance level heterogeneity, performance in query translation and results retrieval, complexity of the mapping process and complex query constraints satisfaction. In our group we have develop different models and tools to deal with many of these issues. In this section we present the query translation and data integration complete method we use to cope with schema level heterogeneities and query constraints satisfaction. The other problems mentioned above are also tackled by our system (see section IV), but the specific methods are beyond the scope of this paper.

This section illustrates the method used to obtain a set of results R from a given query over the global schema Q. The goal is to integrate all partial results from the local sources giving all possible information, while satisfying the constraints contained in Q. The process involves: i) the generation of a set of queries in terms of the local databases integrated in the system, ii) the storage of the partial results that these queries generate into an auxiliary database, and iii) the extraction of the final results from this database. The procedure of translating a global query into a set of local queries has been covered in many previous publications [6, 7, 9, 10, 11, 12, 14, 15]. The approach exposed in this work does not differ from them in the essence of query translation. However, complete support for cross constraints is provided by using an auxiliary database to store partial results. In [21], an auxiliary database is employed to store partial results, but only relational databases are supported, and only join constraints are applied over the auxiliary database. By contrast, our system integrates RDF-based databases—which encloses the relational model—and support any kind of constraint included in the global query, even when in involves data from different sources.

Figure 1 depicts the schema of the complete process of answering global queries.

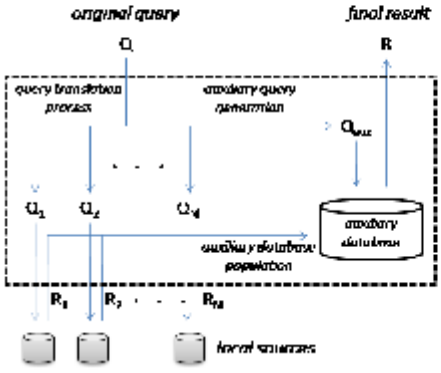

Fig 1. General schema of the database integration system.

The next subsections describe the process to obtain the set of local subqueries and the auxiliary query from a given global query, the generation of the auxiliary database for retrieving the final result, and the general algorithm followed by the system.

A. *Translation of global queries*

This subsection contains definitions which are later used to describe the general algorithm of the integration system.

Definition 3.1 A *query* Q is a triple $\{S, V, C\}$, where S is the set of symbols representing the queried variables—in the *SELECT* statement—, V is the set of global views composing the query, and C is the set of constraints contained in the query.

Definition 3.2 A *view combination function* α is a function that generates all possible combinations of views from a given set of views and a set of joining constraints: $\alpha(V, C) \rightarrow \{v_i | \exists vi_1, vi_2 \in V, \exists (var_1 = var_2) \in C, v_i = join\ (vi_1, vi_2)_{(var_1 = var_2)}\}$.

Definition 3.3 A *view translation function* β is the function that translates global views into semantically equivalent local views—as defined in a given local description: $\beta(ld, vi) \rightarrow v | (v \approx vi) \in ld$, meaning $(v \approx vi)$ that v and vi are mapped views in ld

Definition 3.4 A *query translation function* γ is the function that generates all local queries and the query for the auxiliary database from a given global query and a set of local descriptions LD: $\gamma(S, V, C, LD) \rightarrow \{LQ, Q_{aux}\}$. The generated elements are LQ—the set of local queries—and Q_{aux}—the query for the auxiliary database.

LQ is itself composed by three sets of elements: $LQ = \{S'_i, V'_i, C'_i | 1 \leq i \leq N_l\}$, being N_l is the number of local descriptions—mappings with local sources. S'_i represents the set of queried variables in local query i, V'_i is the set of views in local query i and finally C'_i is the set of constraints in local query i. We define these as follows: V'_i is the translation of the views in V with ld_i. $V'_i = \{v'_j | \exists vi \in V, \beta(ld_i, vi) = v'_j\}$. C'_i is formed by constraints contained in C whose complete set of variables exist in at least one of the views of the local query: $C'_i = \{c'_j | c'_j \in C, \forall var \in c'_j\ \exists v'_k \mid var \in v'_k\}$. These are the necessary constraints for correctly performing query i. Finally, the queried variables in the local query S'_i are in first place the intersection of the queried variables in the global query with the variables contained in the set of local views $S \cap varsInV'_i$ —of course global variables are previously translated according the local descriptions. In addition, variables from cross constraints—those that affect variables from more than one local source—must be queried too, since those constraints are treated a posteriori $\{var_j | \exists c \in C, \exists var_k, \exists V'_i, var_j, var_k \in c,\ var_j \in V'_i, var_k \in V'_i\}$.

The other product of function γ is Q_{aux} which, again, is composed by a set of queried variables, a set of views, and a set of constraints: $Q_{aux} = \{S, V_{aux}, C_{aux}\}$. The queried variables are the same as in the original query, as this query must provide the result for that query. The views in Q_{aux} are

the projection of the variables in S in S'_i: $V = \{\Pi_{S'_i}(W) | 1 \leq i \leq N_l\}$, where W is the table containing all the variables from the original query. Finally, C_{aux} is the set of cross constraints, that is, constrains which affect views from different local queries: $C_{aux} = \{c_l | c_l \in C, \exists var_1, \exists var_2, \exists V_1 \in V'_j, \exists V_2 \in V'_k, var_1, var_2 \in c_l, var_1 \in V_1, var_2 \in V_2\}$.

Therefore, given a global query Q posed by the user, the process of translating Q to generate a set of subqueries expressed in terms of the underlying databases is first applying function a—to generate all combinations of global views—an function c—to generate the set of local queries. This process also produces the auxiliary query, used to retrieve the final results from the auxiliary database.

B. *Generation of the auxiliary database*

The RDF-based auxiliary database is populated with the results retrieved from the local sources. The purpose is to obtain the result to the original query from it. The data stored in it is subsequently retrieved with an auxiliary query—which details were given in the previous subsection—, enabling the use of cross constraints.

First, the RDF Schema for this database is created. For each local source queried, a class representing one table of results is created—we will call this table class. Then, one class is created for each variable in the results of a local query—we will call this variable class. These classes are provided with a datatype property to store their actual results. The class representing the table of results is linked to the classes representing its variable by means of object properties. Figure 2 depicts the relation between one table of results from a local source and the RDF Schema of the auxiliary database.

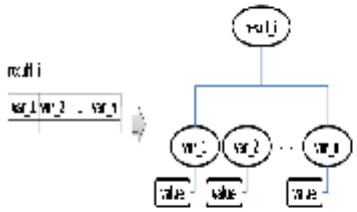

Fig 2. Generating the RDF Schema of the auxiliary database from the result of one local source.

When the RDF Schema is complete, the database is populated with the actual results. The procedure is as follows: for each row of results in a table, one instance of its table class and one instance of each of its variable classes are defined. The values in the row are fed into the datatype properties of the corresponding variable classes. This way, values belonging to the same row are related with each other.

This procedure is repeated for all results obtained from local sources. The generated auxiliary query is able to retrieve and correctly integrate the partial results, producing a single result which corresponds to the original query.

C. *3.3 General algorithm*

Having described all elements and functions that take part in the process of answering queries posed in the system, we can now define the general algorithm followed by the system to obtain a result from a global query.

*Given **Q**: a global query posed in the system, composed by **S**—the set of queried variables—, **V**—the set of views composing **Q**—and, **C**—a set of constraints over variables in **V**.*

*Given **LD**: the set of local descriptions containing the mappings between the global schema and the schemas of the local sources.*

1. Apply function α to V and C in order to obtain all possible combinations of views. Use the newly generated set of views in Q instead of V.
2. Apply function γ to Q and LD in order to obtain the set of local queries LQ and the auxiliary query Q_{aux}.
3. For each local query Q_l in LQ:
    a. Launch Q_l against the local source, obtaining the result R_l.
    b. Generate the part of the RDF Schema of the auxiliary database corresponding to R_l.
    c. Populate the auxiliary database with the data rows contained in R_l.
4. Perform query Q_{aux} over the auxiliary database, obtaining the results R.
5. Return R.

IV. TOOLS

The ACGT Semantic Mediation layer is comprised by several tools designed to answer and optimize queries. Database integration can be divided in a range of sub problems that mostly need to be solved using different approaches (see section III). In our system we have developed a range of collaborating services supporting the different aspects of the general task. These services interact with the query and data to obtain consistent results. The architecture of the ACGT Semantic Mediation Layer is shown in figure 3.

As can be seen, the Semantic Mediator acts as the core of this system, enabling interactions with the rest of tools. OntoQueryClean [22] and OntoDataClean [23], two systems developed in our group and reported before, are both devoted to solve instance level heterogeneities. Their services are invoked by the mediator when necessary, using the defined interfaces. Although it is not within the query translation process itself, we have decided to include the

Mapping Tool as part of the system, given the importance of the mapping process in mediation. The mapping tool aids in the construction of mappings between the global schema and the data repositories. By using these mappings, the mediator is able to automatically produce the required local views and restricted global schemas, necessary for the given database integration approach.

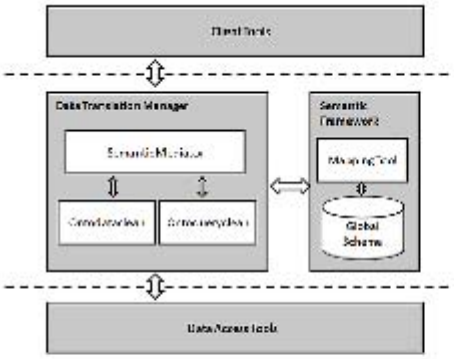

Fig 3: Architecture of the Semantic Mediation Layer

The interfaces used for communicating the services, both internally and externally, are based in web service technologies. An entity acts as a client of a service if it needs something or as a supplier when it is needed. The main internal client of the ACGT Semantic Mediation Layer is the Semantic Mediator, given that it coordinates the work of all the other components. The Semantic Mediator requests, for example, to OntoDataClean to homogenize a set of retrieved data before proceeding with its integration in the general result set.

Data representation in the ACGT Semantic Mediation Layer is based on RDF. We use an RDF Schema (RDFS) to represent the data structure. This global schema represents the possible queries that a client can formulate. The user then gets the perception of querying a unique RDF repository. Regarding the query language, SPARQL [24] was chosen due to its intermediate level of expressiveness and suitability for general purpose applications, as the integration of heterogeneous data sources. A query in SPARQL contains a description of a subset of the global schema in terms of views. Each one of these views has a meaning that is mapped to the corresponding view in one or more underlying databases. The SPARQL query contains a description of the structure of the results and a set of constraints.

This software was built using java technologies, and is exposed as an OGSA-DAI service. The Semantic Mediator is the core tool of this layer. It has two main services: 1) to launch a query, and 2) to browse the schema. The former offers the possibility of sending a query formulated in terms of the global schema for a given integrated set of databases. The Semantic Mediator returns the results in a selected format, together with a metadata file, containing semantic annotations for these data. The second service shows a restricted version of the global schema, representing only the queries that are possible in the integrated database set

Requests are sent to the Semantic Mediator in form of an OGSA-DAI request document [25, 26]. This document contains a very simple workflow, comprised by OGSA-DAI activities. In the case of the Semantic Mediator, this document contains a chain of activities invocations specifying things like the query, in which form the client wants results to be retrieved or where they have to be delivered.

V. Experiments and Results

In this section, we address the issue of proving the feasibility and fitness of the selected approach, together with the performance of the presented tools. We have tested our system with a range of heterogeneous databases of the biomedical domain. Previous experiments, including image (DICOM) and relational sources, with preliminary versions of the system were performed and documented [27]. To achieve this goal, we have classified the different types of conflicts that can be present at different levels.

We identify four conflict categories: a) instance level conflicts, b) schema level conflicts, c) instance-schema conflicts and d) cross constraint conflicts. Instance level conflicts can be present in the retrieved data or in the original query. These conflicts are treated by OntoDataClean and OntoQueryClean tools. Experiments in this area have been documented in previous works [23, 24]. Schema level and instance-schema level conflicts are registered in the mapping files and solved by the mediation algorithm. A schema level conflict is a semantic heterogeneity among two or more databases related to elements in their schemas. When a schema element in the global schema needs to be mapped to instance level knowledge in an underlying database, we say that we have an instance-schema conflict. These conflicts are solved at the mapping level as well. Finally, we find a cross constraint conflict when the original query contains a restriction that involves information from different databases.

In our experiments, we have selected as global schema the semantic core of the ACGT platform: the ACGT Master Ontology on Cancer (MO) [28]. This ontology covers the domain of clinical trials on cancer, so we built ad-hoc databases in this field presenting the different cases of heterogeneity included on purpose. This way, we knew the expected results for every query, being able to evaluate the behavior of the system.

A. Case Study

This subsection presents in detail the execution of one of the tested queries, explaining how the system acts in each of the steps and what products are obtained from it. Figure 4 shows the global query performed in the system.

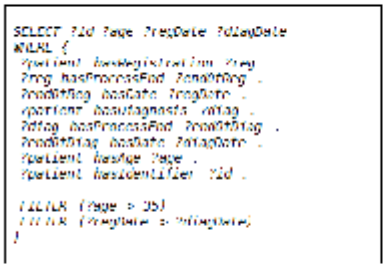

Fig 4: Global query involving data from two repositories

This query retrieves data contained in two different repositories. Namely, the patient's personal data, including age, and the date of registration are stored separately from the date of diagnosis. Data from these sources is related by a patient identifier field. From the two constraints contained in the query, the one involving only the age of the patients can be dealt directly in the sources. The second constraint however involves data from more than one source, thus must be tackled a posteriori—it will be appropriately included in the auxiliary query.

The system identifies the elements contained in the query and applies the mapping information in order to generate the queries for the local repositories. The query for the first repository retrieves the patient's id, , age and registration date, properly restricting for values of age greater than 35. The query for the second repository gathers patient identifiers together with dates of diagnosis. The results obtained from these two queries are partially shown in figure 5.

id	*age*	*reg_date*
1	45	12-03-2005
2	74	23 03 2006
3	36	04 07 2005
4	67	11-03-2005
5	51	14-12-2005
…	…	…

id	*diag_date*
1	01-03-2005
2	29 12 2005
3	29 06 2005
4	03-04-2005
5	20 05 2005
…	…

Fig 5: Part of the result sets obtained with the generated subqueries

These result sets allow the system to build the auxiliary database described in section III. This RDF database stores all the results from the local sources, without performing any kind of join operation. In this specific experiment, two classes—one for each result set—are created. The first one is related to other four classes representing the three fields contained in the corresponding result. Similarly, the second class is related to two more classes, representing the *id* and *registration_date* fields respectively. Parallel to the population of this database, the system automatically generates the auxiliary query, devoted to extracting the final results from the auxiliary database. Figure 6 shows the auxiliary query generated for this experiment.

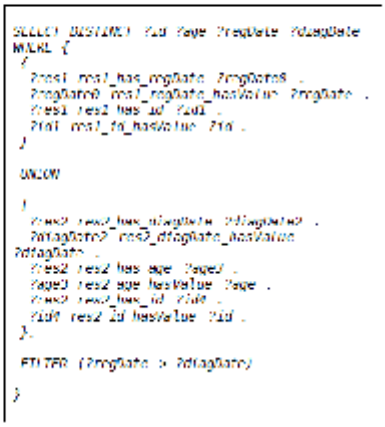

Fig 6: auxiliary query generated by the system

This query includes the second constraint contained in the original query, since it could not be treated by the individual sources. This way, we ensure that the results obtained from this query represent indeed the results corresponding to the global query posed by the user. Figure 7 shows the results retrieved from the auxiliary database.

id	*age*	*reg_date*	*diag_date*
1	45	12-03-2005	01-03-2005
2	74	23-03-2006	29-12-2005
3	36	04-07-2005	29-06-2005
4	67	11-03-2005	03-04-2005
5	51	14-12-2005	20-05-2005
…	…	…	…

Fig 7: final results retrieved from the auxiliary query and returned by the system

As can be seen, the results contain the product of a proper merging between the partial results of the two involved local repositories. The identifier field was used to perform the join of similar data. The cross constraint was successfully processed, and no unexpected data was included in the final results.

B. Results

We built two integrated repositories presenting different types of heterogeneities using the databases mentioned above. A set of queries were designed to test the behavior of the system in different scenarios. These queries tested not only isolated heterogeneities, but also cases of combination. The system was running in a single Intel Core 2 Duo 3Gb+, 8Gb RAM computer. A set of 20 queries were built to test the different cases of heterogeneity present in both repositories.

In all cases, heterogeneities were solved and constraints were satisfied. The automatic construction and storage of the temporal repository did not lead to complexity problems. However, no formal tests regarding performance were done. We understand that the creation of the temporal repository is not more complex than unifying separate results directly. We plan to study performance issues in future studies.

VI. CONCLUSION

Classical Local as View database integration approaches are based on union operations to join the individual results and build an integrated data set. In this work we present a method uses a temporal RDF data repository to allow more complex constraints in the original query, involving data from different data sources. In future releases of our system, we plan to include features such as aggregation operators. Although aggregation is not yet supported by SPARQL, this type of specifications could be easily added to the query system, and applied directly to the temporal repository.

ACKNOWLEDGMENT

We would like to thank all partners in the ACGT project for their technical advisory. We want to thank also Prof. Victor Maojo for his comments on this work.

REFERENCES

[1] J. D. Ullman, "Information integration using logical views", in Proceedings of the International Conference on Database Theory (Delphi, Greece), 1997, pp 19–40.

[2] A.Y. Levy, A. Rajaraman, and J.J. Ordille, "Querying heterogeneous information sources using source descriptions", in Proceedings of the Twenty-second International Conference on Very Large Data Bases (VLDB'96), Mumbai (Bombay), India, September 1996, pp. 251–262.

[3] D. Perez-Rey, V. Maojo, M. Garcia-Remesal, R. Alonso-Calvo, H. Billhardt, F. Martin-Sánchez and A. Sousa, "ONTOFUSION: Ontology-based integration of genomic and clinical databases", in Computers in Biology and Medicine, In Press, Corrected Proof, Available online 6 September 2005

[4] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J Widom, "The TSIMMIS project: Integration of heterogeneous information sources", in Proceedings of the 10th Meeting of the Information Processing Society of Japan, , Tokyo, Japan, October 1994, pp. 7–18.

[5] Bozsak E. et al. "KAON - Towards a Large Scale Semantic Web". In K. Bauknecht, A. Min Tjoa, and G. Quirchmayr, editors, EC-Web 2002, volume 2455 of Lecture Notes in Computer Science, Springer, September 2002, pp. 304–313.

[6] [Ullman00] J.D. Ullman, "Information integration using logical views", Proc. of the 6th Int. Conf. on Database Theory (ICDT'97), volume 1186 of Lecture Notes in Computer Science, Springer, 1997, pp. 19-40.

[7] L. Xu, and D.W. Embley, "Combining the Best of Global-as-View and Local-as-View for Data Integration", Proceedings of ISTA 2004: 3rd International Conference on Information Systems Technology and its Applications, Salt Lake City, USA, 2004, pp. 123-136.

[8] N.T.H. Saw, and K.H.S. Hla, "Semantic Interoperating and Accessing Heterogeneous and Autonomous XML Sources", Proceedings of the 2006 IEEE/WIC/ACM international conference on Web Intelligence and Intelligent Agent Technology (WI-IAT'06), Hong Kong, 2006, pp. 216-219.

[9] A.Y. Halevy, "Answering queries using views: A survey", VLDB Journal, 10:4, 2001, pp. 270-294.

[10] R. Pottinger, and A. Halevy, "Minicon: a scalable algorithm for answering queries using views", VLDB J. 10(2), 2001, pp. 182-198.

[11] A. Calì, D. Calvanese, G. De Giacomo, M. Lenzerini, P. Naggar, and F. Vernacotola, "IBIS: Semantic data integration at work", Proc. of the 15th Int. Conf. on Advanced Information Systems Engineering (CAiSE 2003), volume 2681 of Lecture Notes in Computer Science, Springer, 2003, pp. 79-94.

[12] H. Xiao, and I.F. Cruz, "Integrating and Exchanging XML Data Using Ontologies", LNCS Journal on Data Semantics, Springer Verlag, 2006, pp. 67-89.

[13] P. Lehti, and P. Fankhauser, "XML Data Integration with OWL: Experiences and Challenges", 2004 Symposium on Applications and the Internet (SAINT 2004), 2004, pp. 160-170.

[14] S.D. Camillo, C.A. Heuser, and R. dos Santos Mello, "Querying Heterogeneous XML Sources through a Conceptual Schema", Proceedings of the 22nd International Conference on Conceptual Modeling (ER 2003), 2003, pp. 186-199.

[15] M. Lenzerini, "Data integration: A theoretical perspective", Proceedings of the Symposium on Principles of Database Systems (PODS), 2002, pp. 233-246.

[16] J. Lin, and A.O. Mendelzon, "Merging databases under constraints", Int. J. of Cooperative Information Systems, 7(1), 1998, pp. 55-76.

[17] D. Lembo, M. Lenzerini, and R. Rosati, "Source inconsistency and incompleteness in data integration", Proc. of the 9th Int. Workshop on Knowledge Representation meets Databases (KRDB 2002), 2002.

[18] B. Amann, C. Beeri, I. Fundulaki, and M. Scholl, "Ontology-Based Integration of XML Web Resources", Proceedings of the 1st International Semantic Web Conference (ISWC 2002), 2002, pp. 117-131.

[19] L. Jian, and J. Beihong, "Query Division and Reformulation in Ontology-Based Heterogeneous Information Integration", 15th International Conference on Computing (CIC '06), Mexico City, Nov. 2006, pp. 186-196.

[20] A. Tomasic, L. Raschid, and Patrick Valduriez, "Scaling Access to Heterogeneous Data Sources with DISCO", IEEE Transactions on Knowledge and Data Engineering, v.10 n.5, September 1998, pp.808-823.

[21] E. Mena, V. Kashyap, A.P. Sheth, and A. Illarramendi, "OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies", Proceedings of the 1st IFCIS International Conference on Cooperative Information Systems (CoopIS 1996), 1996, pp. 14-25.

[22] A. Anguita, L. Martin, J. Crespo, and M. Tsiknakis, "An Ontology Based Method to Solve Query Identifier Heterogeneity in Post-Genomic Clinical Trials", Proceedings of the 21st International Congress of the European Federation for Medical Informatics (MIE2008), Gotoborg (Sweden), May 25-28, 2008, pp. 3-8.

[23] D. Perez-Rey, A. Anguita, and J. Crespo, "OntoDataClean: Ontology-based Integration and Preprocessing of Distributed Data", Lec. notes in Computer Science 4345, 2006, pp. 262-272.

[24] [SPARQL] SPARQL Query Language for RDF. Available at: http://www.w3.org/TR/rdf-sparql-query/

[25] The OGSA-DAI Project. Available at: http://www.ogsadai.org.uk/

[26] M. Antonioletti, M.P. Atkinson, R. Baxter, A. Borley, N.P. Chue Hong, B. Collins, N. Hardman, A. Hume, A. Knox, M. Jackson, A. Krause, S. Laws, J. Magowan, N.W. Paton, D. Pearson, T. Sugden, P. Watson, and M. Westhead. "The Design and Implementation of Grid Database Services in OGSA-DAI". Concurrency and Computation: Practice and Experience, Volume 17, Issue 2-4, February 2005, pp. 357-376.

[27] L. Martin, E. Bonsma, A. Anguita, J. Vrijnsen, M. Garcia-Remesal, J. Crespo, M. Tsiknakis, V. Maojo, "Data Access and Management in ACGT: Tools to Solve Syntactic and Semantic Heterogeneities Between Clinical and Image Databases", in Advances in Conceptual Modeling – Foundations and Applications, Lecture Notes in Computer Science, 2007, pp. 24-33.

[28] M. Brochhausen, G. Weiler, C. Cocos, H. Stenzhorn, N. Graf, M. Doerr, M. Tsiknakis, "The ACGT Master Ontology on Cancer - a New Terminology Source for Oncological Practice", in IEEE CBMS 2008: 21st IEEE International Symposium on Computer-Based Medical Systems, Jyväskylä, Finland, June 17-19, 2008.