UNIVERSIDADE DA CORUÑA

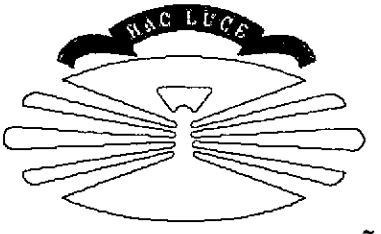
Departamento de Tecnoloxías da Información e as Comunicacións

TESIS DOCTORAL

UN SISTEMA MEDIADOR PARA LA INTEGRACIÓN DE DATOS ESTRUCTURADOS Y SEMI-ESTRUCTURADOS

Carlos Alberto Paz Bermúdez

Director: Dr. Angel Viña Castiñeiras



UNIVERSIDADE DA CORUÑA

Departamento de Tecnoloxías da Información e as Comunicacións

TESIS DOCTORAL

UN SISTEMA MEDIADOR PARA LA INTEGRACIÓN DE DATOS ESTRUCTURADOS Y SEMI-ESTRUCTURADOS

Carlos Alberto Pan Bermúdez

Director: Dr. Angel Viña Castiñeiras

A mi familia
A Silvia
A mis amigos

AGRADECIMIENTOS

En primer lugar, quiero agradecer a mi tutor Ángel Viña el haberme proporcionado el enfoque y los medios necesarios para llevar a cabo este trabajo.

También debo reconocer mi deuda con la gente, que desde el Departamento de Tecnologías de la Información y las Comunicaciones de la Universidad de A Coruña, y desde el departamento de I+D de DENODO Technologies, han trabajado en la implementación real de estas técnicas en un sistema funcional. Quiero mencionar especialmente a Manuel Álvarez y Juan Raposo, que han dirigido conmigo las labores de implementación del sistema, y que han hecho innumerables sugerencias útiles. También han participado en el desarrollo: Paula Montoto, Vicente Orjales, Lucía Ardao, Salvador Fandiño, Lorena Martínez y Daniel Castro. Sin ellos, este trabajo sería muy diferente.

Los desarrolladores que han utilizado la implementación de estas técnicas para la construcción de aplicaciones industriales también me han ayudado con sus ideas y sugerencias. No me es posible citarlos a todos, pero su aportación también ha sido muy productiva para este trabajo.

He recibido otras colaboraciones muy valiosas. David Sánchez y Justo Hidalgo me ayudaron a revisar varios de los algoritmos y técnicas aquí presentados. Anastasio Molano me dio consejos muy útiles sobre la estructura de esta memoria y la manera de enfocar la discusión del trabajo relacionado. Víctor Carneiro se preocupó por hacerme fácil la tediosa tramitación de una tesis doctoral.

No puedo olvidarme aquí del resto de mis compañeros en DENODO Technologies (los que estuvieron y los que siguen estando) y en el TIC, ni de la gente con la que he trabajado en el Área de Telemática de la Universidad Carlos III de Madrid. Todos ellos han influenciado mi trabajo de una u otra manera.

Por último, no hubiera podido sacar adelante este trabajo sin el apoyo constante de mi familia, de Silvia y de mis amigos.

A todos ellos, gracias.

RESUMEN

El mundo actual se caracteriza por la enorme abundancia de información y por el fácil acceso a la misma. Sin embargo, esta información es difícil de manejar adecuadamente, ya que muy a menudo se encuentra dispersa, es heterogénea y presenta un nivel de estructuración bajo.

La llamada "telaraña mundial" (o World Wide Web) es un perfecto ejemplo de este fenómeno. Convertida ya en el mayor repositorio de información de la historia, su naturaleza abierta ha llevado, sin embargo, a que sus contenidos estén débilmente estructurados y, cuando lo están, su esquema sea enormemente heterogéneo. Se podría resumir la situación diciendo que el WWW es un repositorio de información *legible para las personas* pero que no es *legible para máquinas*, lo cual impide su procesamiento automático. Una situación similar, a menor escala, ocurre en los sistemas de información de entornos corporativos medianos-grandes.

Los sistemas mediadores pretenden proporcionar a sus usuarios una visión unificada sobre fuentes de datos dispersas, heterogéneas y, posiblemente, débilmente estructuradas. En este enfoque, los datos permanecen en las fuentes originales y el *mediador* es responsable de proporcionar a sus usuarios la "ilusión" de estar consultando una única fuente de datos con un esquema global único y coherente.

El objetivo principal de esta tesis doctoral es la construcción de un sistema mediador que reúna todas las características necesarias para su utilización en la construcción de aplicaciones de integración de datos en entornos de producción reales. Si bien el alcance de su aplicabilidad es mayor, los objetivos de este sistema hacen un énfasis especial en proporcionar soporte a dos ámbitos concretos de aplicación: aplicaciones de búsqueda/agregación en Internet por un lado, y aplicaciones de apoyo al negocio electrónico y la gestión del conocimiento dentro de organizaciones medianas o grandes, por otro.

Entre las principales contribuciones de este trabajo se encuentran: un algoritmo para calcular las capacidades de consulta del esquema global en función de las de las fuentes, que es capaz de utilizar un modelo de representación de capacidades de consulta más rico que en sistemas anteriores; un sistema semi-automático para la generación de envoltorios para fuentes web que puede ser utilizado por no programadores; y un esquema que permite escoger de manera flexible la estrategia de materialización a utilizar en cada aplicación.

ABSTRACT

The world today is characterised by the easy access to great volumes of information. Nevertheless, it is very difficult to properly manage these information, since it is disperse, heterogeneous and weakly structured.

One of the best examples of this situation is the World Wide Web. Having become the largest data repository in the whole history, its openness has lead it to a state where its contents are weakly structured and are very heterogeneous. We could summarize it by saying that WWW is *human-readable* but it is not *machine-readable*. A similar situation can be found in many medium-large corporate information systems of today.

Mediator systems aim to provide users with an unified view over disperse, possibly weakly structured, and heterogeneous data sources. In this approach, the data remains in the sources and the *mediator* is responsible for providing users with the "illusion" of being querying a single and coherent global schema

The main goal of this PhD. thesis is building a mediator system able of dealing with the complexities of real-world industrial applications. Though it can be used in other domains, our objectives are specially focused over two application domains: on one hand comparison, aggregation and vertical search applications in the Internet; on the other hand, applications for supporting *e-business* and *knowledge management* in corporate environments.

Some of the main contributions of this work follows: an algorithm for computing global schema query capability from sources, which can deal with a more advanced query capability representation framework than previous systems; a semi-automatic wrapper generation system for web sources, which can be used by non-programmers; and a cache system which let administrators define to suit the materialization schema used for each application.

Índice

I. PLANTEAMIENTO Y CONTRIBUCIONES

I.1. ÁMBITO

Esta tesis doctoral aborda el problema de la Integración de Datos Distribuidos. Este problema consiste en combinar y unificar fuentes de datos conectadas a través de una red, que han sido creadas de manera independiente y que, por lo tanto, no tienen por qué seguir esquemas ni convenciones de representación comunes o incluso compatibles.

El problema de Integración de Datos Distribuidos ha recibido una atención creciente en los últimos años. El mundo actual se caracteriza por la enorme abundancia de información y por el fácil acceso a la misma. Sin embargo, esta información se encuentra dispersa y ha sido, y sigue siendo, generada de manera independiente y siguiendo muy diversos esquemas y formatos. Esto complica sobremanera su combinación y gestión unificada. Por otro lado, gran parte de la información generada se encuentra en formatos entendibles por sus usuarios humanos pero no adecuados para su procesamiento automático.

Esta es la causa de que el problema de Integración de Datos haya sido abordado de manera muy activa por investigadores de diferentes comunidades y disciplinas, tales como la Telemática, las Bases de Datos, la Inteligencia Artificial, la Recuperación de la Información o la Minería de Datos, realizándose en los últimos años importantes contribuciones desde diferentes puntos de vista.

Dentro del problema general de la Integración de Datos Distribuidos, las principales técnicas desarrolladas pueden dividirse en función de las características de estructuración de las fuentes que son capaces de abordar.

Tradicionalmente se ha distinguido entre datos estructurados y datos no estructurados. Los datos no estructurados son aquellos que no presentan ningún esquema más allá de la mera secuencia de bytes o palabras. Los datos estructurados son aquellos que siguen un esquema perfectamente definido (e.g. aquellos contenidos en una base de datos).

Sin embargo, en los últimos años ha cobrado especial importancia una clase de datos que no puede encuadrarse adecuadamente en ninguna de las dos categorías anteriores. Esta nueva categoría de datos ha recibido el nombre de datos semi-estructurados[Abi97]. Los datos semi-estructurados se caracterizan por seguir algún tipo de esquema, pero de características mucho menos rígidas que las encontradas en los datos estructurados. De hecho, el esquema puede incluso no estar descrito separadamente de los datos, sino encontrarse implícito en los mismos.

Si bien es posible encontrar abundantes ejemplos de datos con estas características dentro de cualquier ámbito, quizás haya sido la multitud de fuentes de esta clase disponibles dentro de la llamada telaraña mundial (World Wide Web), la que en mayor medida ha desencadenado el interés por el aprovechamiento de sus regularidades implícitas para permitir un tratamiento más flexible y automatizado sobre ellos. También la abundancia de fuentes de este tipo dentro de los entornos corporativos de tamaño mediano o grande, ha constituido un importante acicate.

Un foco importante de investigación en la actualidad es la construcción de sistemas capaces de integrar de manera sencilla datos estructurados y semi-estructurados heterogéneos y dispersos en múltiples fuentes accesibles por medios telemáticos, de manera que se obtenga de los mismos una visión similar a la proporcionada por una base de datos convencional.

Quizás la aproximación arquitectural más utilizada hasta el momento para la construcción de este tipo de sistemas, sea la de mediador[Wie92]. En ella, los datos permanecen en sus fuentes originales y un sistema intermedio (llamado mediador), se encarga de proporcionar a los usuarios la ilusión de que existe una única fuente en la que se encuentran todos los datos combinados y unificados de manera coherente de acuerdo a un único esquema global. Cuando el mediador recibe una consulta sobre el esquema global, la reformula en diversas subconsultas sobre las fuentes originales. La interacción directa con las fuentes es delegada por el mediador a los llamados programas envoltorio (o *wrappers*). Estos se encargan de recibir las subconsultas enviadas por el mediador, traducirlas al formato utilizado por la fuente, ejecutarlas sobre la misma y obtener los resultados, devolviéndoselos al mediador. Entonces, los resultados obtenidos de las fuentes son reestructurados por el mediador para ajustarse al esquema global y devueltos al usuario. De esta manera, éste obtiene la impresión de estar consultando un único sistema.

La construcción de sistemas mediadores plantea importantes retos tales como la reformulación de las consultas sobre el esquema global en subconsultas sobre las fuentes o el proporcionar mecanismos que permitan generar y mantener de manera sencilla los programas envoltorio (algo fundamental si se desea que las aplicaciones construidas con el sistema puedan escalar adecuadamente en cuanto al número de fuentes).

El objetivo principal de esta tesis doctoral es la construcción de un sistema mediador que reúna todas las características necesarias para su utilización en la construcción de aplicaciones de integración de datos en entornos de producción reales. Si bien el alcance de su aplicabilidad es mayor, los objetivos de este sistema hacen un énfasis especial en proporcionar soporte a dos ámbitos concretos de aplicación, que han levantado importantes expectativas industriales sobre las técnicas avanzadas de integración de datos distribuidos: aplicaciones de búsqueda/agregación en Internet por un lado, y aplicaciones de apoyo al negocio electrónico y la gestión del conocimiento dentro de organizaciones medianas o grandes, por otro.

I.1.1. APLICACIONES DE COMPARACIÓN / BÚSQUEDA / AGREGACIÓN EN INTERNET

En los últimos años han aparecido multitud de sitios web que, con propósitos comerciales o sin ellos, proporcionan algún tipo de servicio a los usuarios de Internet.

Por ejemplo, son abundantes las tiendas electrónicas o sitios web que venden todo tipo de productos. También han alcanzado gran popularidad los sitios web de empleo, donde se ofrece a las empresas un medio barato y de alta audiencia para publicar ofertas de empleo y pre-seleccionar personal, mientras a los particulares se les proporciona una manera cómoda, rápida y sencilla de localizar ofertas adecuadas a su perfil y presentar su candidatura a las mismas.

Otro ejemplo lo constituyen servicios bancarios y financieros, tales como las bancas electrónicas ofrecidas a sus clientes por, prácticamente, todas las entidades bancarias.

La lista de ejemplos podría ser, en realidad, casi infinita: periódicos y medios de comunicación, subastas de bienes y servicios, sitios web de información especializada en cualquier ámbito, etc.

Sin embargo, en la amplia mayoría de casos, podemos encontrar una situación común: el usuario encuentra que los servicios que desea están distribuidos en diferentes sitios web y no resulta sencillo combinarlos o unificarlos en aras de compararlos, obtener una mayor potencia o una superior comodidad de uso.

Dado que la mayoría de estas fuentes web proporcionan datos que cumplen los requisitos que son exigidos a los datos semi-estructurados, los sistemas mediadores pueden ser aplicados en este ámbito. Además, aportan otra serie de ventajas frente a otros enfoques, como el de Almacén de Datos (o *Data Warehouse*). Ver apartado II.2.2), que son primordiales en este entorno concreto: permiten la consulta de los datos en tiempo real (esto es, actualizados), escalan mucho mejor en cuanto al número de fuentes, evitan la transferencia periódica de enormes volúmenes de datos y pueden funcionar incluso cuando el grado de autonomía de las fuentes es total.

I.1.2. APLICACIONES DE INTEGRACIÓN DE INFORMACIÓN EN ENTORNOS CORPORATIVOS

Las organizaciones que operan en el mundo contemporáneo se encuentran con nuevos retos derivados de los cambios que se han experimentado recientemente en el entorno de negocio mundial. Las organizaciones se encuentran, por un lado con que su operativa genera una cantidad de información enorme, que es difícil de gestionar y a la que es complicado, cuando no imposible, sacar el máximo partido. Por otro lado, factores tales como la rápida evolución tecnológica, las presiones del mercado o el efecto de la competencia entre diferentes proveedores de soluciones técnicas, ha causado que los entornos y tecnologías utilizadas para la generación y manejo de estos datos difieran en gran medida a lo largo del tiempo, creando así un escenario donde

grandes volúmenes de información dispersa y heterogénea debe ser combinada y unificada.

En la actualidad, la mayoría de organizaciones de tamaño mediano-grande están abordando, o piensan abordar, programas de algunos de los tipos siguientes: Gestión de las Relaciones con los Clientes (*CRM*), Gestión del Conocimiento (*Knowledge Management*), Integración de Aplicaciones (*Enterprise Application Integration*), Portales Corporativos (Enterprise Information Portals), Comercio electrónico Negocio a Negocio (*Business to Business Electronic Commerce*), etc.

Todos estos términos, que serán abordados en mayor detalle en apartados posteriores, tienen en común la necesidad de trabajar sobre visiones unificadas de los datos manejados por la organización. El enfoque utilizado mayoritariamente hasta la fecha se basaba en la idea de la construcción de algún tipo de almacén de datos donde se cargan periódicamente grandes volúmenes de datos unificados, que son utilizados como base para procesos posteriores.

Sin embargo, este enfoque, si bien adecuado para la realización de procesos batch tales como aquellos que requieren algún tipo de Minería de Datos sobre históricos, muestra un importante número de limitaciones tales como: 1) Debido a que los datos no se encuentran actualizados, no es adecuado para aplicaciones que requieran funcionamiento en tiempo real (tales como las que son frecuentes en, por ejemplo, la disciplina de Integración de Aplicaciones) 2) Los sistemas construidos hasta el momento no tienen capacidad para aprovechar las regularidades presentes en los datos semi-estructurados, y tienen que tratarlos como si estuviesen carentes de estructura 3) Debido a que requiere de la construcción de un gran repositorio de datos y de los procesos de migración de datos desde las fuentes originales, su puesta en marcha es larga y costosa 4) No es valido en la práctica para soportar repositorios inter-organización como aquellos necesarios en aplicaciones de Comercio Electrónico de Negocio a Negocio.

Los sistemas mediadores pueden aportar un nivel de flexibilidad mucho mayor: pueden ponerse en marcha de manera más rápida, no son necesarios grandes repositorios centrales, ni diseñar complicados procesos de migración y actualización de datos, permiten tratar los datos semi-estructurados de una manera similar a si fuesen estructurados, permiten la operación en tiempo real, etc.

Comparadas con las aplicaciones en entorno Internet, las aplicaciones en entornos corporativos suelen caracterizarse por un número de fuentes menor y un grado de autonomía de las mismas muy inferior.

I.2. OBJETIVOS

El principal objetivo de esta tesis doctoral es la construcción de un sistema mediador que permita de manera sencilla la integración de fuentes estructuradas y semi-estructuradas, dispersas y heterogéneas. Este trabajo pretende demostrar, mediante su construcción, que es posible crear un sistema mediador que cumpla las siguientes características:

1) Las aplicaciones construidas con el sistema mediador deben ser capaces de funcionar en entornos de producción reales, soportando las cargas de trabajo que se producen en dichos entornos. En particular, debe considerar las necesidades de aplicaciones Internet accesibles a una audiencia multitudinaria, así como las que son generadas por una aplicación utilizada de manera intensiva en una gran corporación para propósitos de negocio. Esto involucra que el sistema debe utilizar una arquitectura distribuida que permita escalar fácilmente la arquitectura hardware y de comunicaciones necesaria.

2) El sistema mediador debe ser capaz de permitir la integración de todos los tipos de fuentes estructuradas y semi-estructuradas habituales tanto en la construcción de aplicaciones Internet como en la construcción de aplicaciones corporativas de integración de datos. Esto incluye, entre otras, fuentes web con su salida en HTML, documentos XML, bases de datos relacionales, hojas de cálculo y documentos de texto semi-estructurado.
Esto involucra que los modelos utilizados deben ser lo suficientemente expresivos para reflejar las peculiaridades de cada tipo de fuente en aspectos como: capacidades de consulta, formatos de representación, mecanismos de ejecución, etc.

3) El sistema mediador debe conseguir proporcionar sobre los datos semi-estructurados una visión que permita manejarlos de manera similar a si estuviesen contenidos en una base de datos convencional (i.e si fuesen datos estructurados).

4) El sistema mediador debe poder ser integrado de manera sencilla en los entornos de producción habituales actualmente en la mayor parte de centros de procesamiento de datos. Esto involucra que el sistema debe ser acorde con las principales normas existentes actualmente en la industria para el acceso y manejo de datos, tales como SQL o JDBC.

5) La creación del esquema global unificado de una aplicación construida con el sistema mediador, así como la configuración del mismo, debe poder ser realizada de manera sencilla por cualquier persona con conocimientos básicos de administración de bases de datos convencionales.

6) El mantenimiento de los envoltorios utilizados, incluidos los de las fuentes web, para las fuentes utilizadas por las aplicaciones generadas con el sistema mediador debe poder ser realizado, en un porcentaje superior al 90%, por personal sin capacidades de

programación. Esto incluye la generación de envoltorios para nuevas fuentes, así como el mantenimiento de los envoltorios para fuentes ya existentes.

7) Los tiempos de generación y mantenimiento de programas envoltorio para fuentes web deben ser cortos. Este trabajo se plantea el objetivo de que el esfuerzo medio de creación de un programa envoltorio para una fuente, incluyendo las fuentes web, sea inferior a 1 día de trabajo de una persona sin habilidades de programación. Esto permite tiempos muy cortos para la creación y puesta en marcha de aplicaciones de integración de datos.

8) El sistema debe ser capaz de funcionar consultando las fuentes en tiempo real, devolviendo así los datos totalmente actualizados. Sin embargo, por propósitos de eficiencia, debe permitir también, siempre a discreción del administrador del sistema, guardar caches o copias locales de parte de los datos de las fuentes, para las aplicaciones en las que esto sea aconsejable.

9) El sistema debe proporcionar algún mecanismo suficientemente flexible para tratar con las heterogeneidades de formatos de representación entre fuentes de datos.

I.3. PRINCIPALES CONTRIBUCIONES ORIGINALES

Frente a sistemas previos presentados en la literatura, el sistema mediador presentado en este trabajo presenta las siguientes aportaciones principales:

1) Un modelo de representación de relaciones y sus capacidades de consulta con la suficiente potencia para tratar las fuentes encontradas en aplicaciones reales, permitiendo tener en cuenta importantes características tales como la posibilidad de utilizar operadores arbitrarios o la representación de las características de multiplicidad de las fuentes (esto es, cuantas condiciones de selección sobre un mismo atributo son capaces de ejecutar en una sola consulta). Este modelo es descrito en el apartado III.2.1.

2) Un algoritmo para calcular las capacidades de las relaciones del esquema global en función de las de las fuentes, que soporta un modelo de representación de capacidades de consulta más rico que los algoritmos anteriormente presentados en la literatura. Disponer de este algoritmo permite conocer *a priori* las consultas que pueden ser contestadas por el sistema mediador, además de permitir que un mediador pueda actuar como fuente para otros mediadores. El algoritmo genera además información base que es utilizada por el algoritmo de ejecución de consultas sobre el sistema. Se describe en el apartado III.2.4.

3) Un sistema de generación de programas envoltorio que permite a usuarios no programadores, generar en muy poco tiempo envoltorios para fuentes web, incluyendo las complejas fuentes comerciales de hoy en día. El sistema es descrito en el apartado III.4.

4) La posibilidad de que el administrador escoja para cada aplicación el esquema de almacenamiento que mejor se ajuste a sus características. Los principales esquemas son: *virtual* (en el cuál los datos permanecen en las fuentes), *materializado* (en el cual copias de los datos son guardados localmente por el mediador) o *mixto* (en el que parte de los datos se guardan localmente y parte permanecen en las fuentes. Esto es logrado mediante un sistema cache que utiliza también un novedoso mecanismo para representar sus contenidos. El sistema cache es presentado en el apartado III.2.5.6.

5) Una arquitectura distribuida que le permite escalar fácilmente para el tratamiento de aplicaciones con cargas de trabajo altas. Además, su confianza en normas de amplia difusión en la industria, tales como JDBC, SQL o un modelo de datos basado en el relacional, permite su rápida integración en los actuales entornos de producción. También facilita la formación en el uso del sistema. La arquitectura del sistema se describe en el apartado III.1.

El enfoque de este trabajo centra la descripción y discusión técnica del sistema mediador en estas aportaciones. Otras partes del sistema, tales como el mecanismo de optimización de consultas (apartado III.2.5.4), son descritos a nivel general para

proporcionar el adecuado contexto para los objetivos de este trabajo y por el interés que, al tratarse de una implementación pionera, tiene la experiencia obtenida durante su uso en aplicaciones reales.

Esa es, a nuestro juicio, otra contribución de este trabajo. Los sistemas mediadores presentados hasta el momento en la literatura no han sido utilizados en la construcción de aplicaciones reales (o, si lo han sido, los resultados no han sido trasladados a la comunidad investigadora). El sistema presentado en esta tesis doctoral ha sido utilizado para la construcción de aplicaciones de integración de datos en diversos ámbitos, soportando requerimientos y cargas de trabajo reales. Los principales resultados son presentados en este documento y permiten, además de validar el sistema aquí propuesto, obtener conclusiones relativas a los sistemas mediadores en general y su adecuación a las modernas necesidades de integración de datos. La evaluación experimental del sistema es descrita en el capítulo IV, utilizando fundamentalmente como banco de pruebas diversos servicios en línea dentro del portal Biwe [Biw].

I.4. ESTRUCTURA DE LA TESIS

El Capítulo II, '*Estado del Arte en Sistemas Mediadores*', comienza con una panorámica general de los modernos sistemas de integración de datos. Posteriormente, ofrece una descripción de las motivaciones que han avivado recientemente el interés por la investigación en nuevas técnicas de integración de información distribuida. La parte principal del capítulo la constituye una descripción detallada del estado del arte actual en las técnicas empleadas en el ámbito concreto de la construcción de sistemas mediadores.

El Capítulo III, '*Descripción del Sistema Mediador*', describe en detalle el sistema presentado en este trabajo. Se comienza con una descripción de la arquitectura del sistema y sus principales componentes. Posteriormente se analizan en profundidad los modelos y algoritmos que se utilizan en el nivel lógico. Seguidamente se entra en detalle en los mecanismos para la generación de programas envoltorio. El sistema para la generación de envoltorios para fuentes web y de texto semi-estructurado reciben una especial atención por ser quizás el tipo de fuentes que ofrece más dificultades en este sentido.

El Capítulo IV, '*Experiencia obtenida del uso del sistema: Aplicaciones y Evaluación*', pretende mostrar al sistema mediador aplicado en la práctica. Comienza con la descripción de dos aplicaciones significativas construidas con el sistema, con el objetivo de proporcionar una visión más clara de cómo el sistema puede utilizarse para resolver problemas reales de integración de datos. Posteriormente, se realiza una evaluación del cumplimiento de los objetivos planteados en función de los datos recogidos de diversas aplicaciones construidas con el sistema.

Finalmente, el Capítulo V, '*Discusión, Conclusiones y Líneas de Trabajo Futuro*', comienza discutiendo el sistema presentado en relación al trabajo relacionado que fue introducido en el Capítulo 2, continua exponiendo las conclusiones de la tesis doctoral, y finaliza esbozando las líneas básicas de trabajo futuro del autor.

II. ESTADO DEL ARTE EN SISTEMAS MEDIADORES

En este capítulo se comienza introduciendo el contexto en el que surge la necesidad de los modernos sistemas de integración de datos accesibles por medios telematicos. Posteriormente se introduce una clasificación para los mismos. Tras un breve comentario acerca de los principales tipos de sistemas, y de la descripción de varios tipos de aplicaciones significativos que constituyen quizás la motivación fundamental para la nueva generación de sistemas de integración, nuestra atención se centra en los sistemas mediadores. Se exponen en primer lugar los principales problemas encontrados a la hora de construir un sistema mediador y posteriormente se realiza un estudio en detalle de los principales enfoques y técnicas adoptadas hasta el momento para enfrentarlos.

II.1. INTRODUCCIÓN

Las nuevas tecnologías surgidas en las últimas décadas han permitido la creación, recopilación y publicación de información en volúmenes nunca antes imaginados y en una amplia variedad de formatos. Esta información está disponible a través de un amplio abanico de medios telemáticos.

Sin embargo, este hecho tiene como consecuencia inherente la aparente paradoja de que si bien nunca se ha dispuesto de un acceso tan sencillo a tanta información, pocas veces ha sido tan difícil de manejar. La proliferación de fuentes de información totalmente independientes entre sí, ha llevado a un escenario en que mucha información potencialmente útil para unas necesidades particulares está dispersa y almacenada según esquemas y estructuras de almacenamiento con grandes heterogeneidades. Esto complica sobremanera la integración y procesamiento automático de datos procedentes de diversas fuentes.

La llamada "telaraña mundial" (o World Wide Web, abreviado WWW) es un perfecto ejemplo de este fenómeno. Convertida ya en el mayor repositorio de información de la historia de la humanidad, su naturaleza abierta ha llevado, sin embargo, a que sus contenidos estén débilmente estructurados y, cuando lo están, su esquema sea enormemente heterogéneo. Se podría resumir la situación diciendo que el WWW es un repositorio de información *legible para las personas* pero que no es *legible para máquinas*, lo cual impide su procesamiento automático.

Otro buen ejemplo de esta tendencia se manifiesta en el entorno corporativo de las empresas de tamaño mediano o grande. Si bien podría esperarse que en este caso el nivel de estructuración y homogeneidad fuese mucho mayor, un informe de la consultora Gartner Group[LK01] demuestra que las primeras 2000 compañías de Estados Unidos disponen de media de más de 100 aplicaciones funcionando sobre 15 plataformas diferentes y con 8 arquitecturas de almacenamiento de datos. Casi

siempre, los esquemas de los repositorios de datos de estas aplicaciones son incompatibles y, por tanto, resulta muy costoso realizar y mantener cualquier aplicación que los combine entre sí. Esto se ve fuertemente agravado por la tendencia actual en aplicaciones de negocio electrónico de conectar los sistemas en línea de una organización con los de sus proveedores, clientes y canales de comercialización.

Además, las corporaciones desean cada vez más sacar partido de los grandes volúmenes de información útil disponible en el WWW, de manera que puedan utilizarla de manera similar a la almacenada en una base de datos corporativa local.

Sin embargo, estos contextos de aplicación presentan las suficientes peculiaridades para que la mayoría de las técnicas tradicionales de manejo de datos distribuidos no puedan aplicarse directamente de manera eficaz. Normalmente los repositorios base tienen una estructura débil, modos de acceso limitados y presentan heterogeneidades muy fuertes. El hecho de que estas fuentes estén además dispersas y accesibles a través de una red también debe ser tenido en cuenta.

Es, por tanto, lógico que en los últimos años se haya desarrollado un intenso trabajo de investigación encaminado a diseñar e implementar métodos eficaces para integrar, gestionar y consultar la información contenida en estos repositorios. En este capítulo se realiza un repaso de los principales conceptos y técnicas desarrolladas hasta el momento en este ámbito. En primer lugar, se describen los principales tipos de fuentes de datos en función de su nivel de estructuración. Posteriormente, se ofrece una clasificación de los modernos sistemas de integración de datos surgidos en los últimos tiempos. En tercer lugar, se describen con más detalle algunos tipos de concretos de aplicaciones para las que existe una fuerte demanda, y que precisan o son considerablemente más eficaces y fáciles de crear con la ayuda de sistemas mediadores. Finalmente, la parte principal del capítulo la constituye una descripción detallada del estado del arte actual en las técnicas empleadas en el ámbito concreto de la construcción de sistemas mediadores.

II.2. FUENTES DE INFORMACIÓN

Tal y como ya se ha establecido, se pretende construir sistemas que integren repositorios de datos dispersos y heterogéneos incluso cuando estos presentan un bajo nivel de estructuración.

En este apartado se realiza una clasificación de los repositorios o fuentes de información en función del nivel de estructuración de sus contenidos. Distinguiremos tres clases de fuentes de acuerdo a este criterio:

- Fuentes no estructuradas
- Fuentes estructuradas
- Fuentes semiestructuradas

Los siguientes epígrafes se ocupan, respectivamente, de cada uno de los tres tipos.

II.2.1. FUENTES NO ESTRUCTURADAS

Denominaremos fuentes no estructuradas a aquellas cuyos elementos de datos:

- No pertenecen a ningún tipo de datos (es decir que su modo de representación y sus propiedades no están fijados).
- No presentan ningún tipo de esquema, más allá de una mera secuencia de bytes o caracteres.

Un ejemplo típico son los documentos en texto libre de cualquier tipo. Debido a su falta de estructura, la única manera posible de consultar estos repositorios es mediante consultas *no estructuradas* o *imprecisas*, tales como las búsquedas por palabra clave. Con este tipo de consultas sólo es posible representar expresiones que recuperen aquellos documentos en los que aparece una determinada secuencia o secuencias de caracteres o bytes. Pueden utilizarse además operadores lógicos para combinar expresiones simples.

Este tipo de consultas son sencillas de realizar pero normalmente son incapaces de representar con precisión las necesidades de información de un usuario. Una de las razones de esto es que trabajan con un nivel de granularidad de documento y no de tupla.

El siguiente ejemplo ilustra algunas de las limitaciones que se derivan de esto.

Ejemplo II.2.1.1: Consultas imprecisas

Supongamos una fuente de información no estructurada consistente en una serie de documentos, cada uno de los cuales contiene una lista de referencias bibliográficas.

Supóngase que un usuario desea obtener aquellos libros de la editorial 'Arkham' que traten o bien sobre Java o bien sobre XML. Probablemente, la consulta no estructurada que mejor podría ajustarse a sus necesidades sería alguna similar a: '(java or xml) and Arkham'.

Esta consulta devolverá aquellos documentos que contengan en algún punto la palabra 'Arkham' y, o bien la palabra 'java' o bien la palabra 'xml'.

Esta consulta no producirá exactamente la respuesta deseada debido a varias razones:

- Puede haber documentos que contengan las palabras 'java' o 'xml' en una referencia bibliográfica y que contengan la palabra 'Arkham' en otra referencia diferente. El documento sería devuelto como respuesta, aunque no contendría realmente referencias como las que el usuario busca.
- La consulta devuelve documentos y no referencias bibliográficas. Es decir, incluso aunque el caso anterior no ocurra, el usuario obtendrá documentos donde está contenida alguna referencia válida, pero junto a otras muchas no válidas.
- Si una referencia tuviese en su título o descripción la palabra 'Arkham', la página conteniéndola aparecería en los resultados: no es posible diferenciar los diferentes campos que constituyen un elemento de información.

Además, hay un amplio número de operaciones que el usuario no podrá realizar en su consulta:

- Ninguna operación que presuponga un cierto tipado de datos será posible. Por ejemplo, no podrán incluirse en la consulta operaciones de comparación, tales como escoger sólo aquellas referencias cuyo precio sea inferior a un cierto valor.
- Ninguna operación de ordenación o agregación será posible. Por ejemplo, el usuario no podrá ordenar las referencias alfabéticamente por título o por precio.
- Los resultados de la consulta no pueden ser procesados automáticamente de alguna manera por un programa (para, por ejemplo, realizar la compra automática de las referencias bibliográficas encontradas), ya que al carecer de esquema, no podrá interpretarlos correctamente.

□

Dentro de la disciplina de Recuperación de Información se han desarrollado numerosas técnicas que permiten indexar y gestionar grandes volúmenes de información textual no estructurada y permiten la realización de consultas imprecisas sobre la misma[Bae98].

Por ejemplo, sistemas como Altavista[Alt] o Google[Goo] consideran el WWW como un gran repositorio de información no estructurada y son capaces de construir índices de gran tamaño sobre la misma, permitiendo su consulta de manera eficiente.

Sin embargo, como ya se ha expuesto, esta abstracción no estructurada del WWW es demasiado débil para la mayor parte de aplicaciones y no explota la estructura que, en mayor o menor grado, sí presentan amplias porciones de la información disponible en el WWW.

II.2.2. FUENTES ESTRUCTURADAS

Son aquellas fuentes que presentan un esquema rígido y bien definido para los datos. Un típico ejemplo de fuente estructurada es una base de datos relacional.

En este caso, la fuente presenta un esquema (almacenado en un diccionario de datos), que define la organización interna de los datos y las restricciones a aplicar sobre los mismos (e.g. posibilidad de existencia de valores nulos, rangos de valores, etc.).

Sobre estas fuentes pueden emitirse *consultas estructuradas* o *precisas*. Existen un gran número de lenguajes que permiten realizar consultas estructuradas, de los cuales el más popular es SQL[SQL92].

El procesamiento automatizado de este tipo de fuentes es posible ya que su férrea estructura las hace adecuadas para su tratamiento por programas de ordenador.
Sin embargo, tal y como ya se ha comentado, hay inmensos volúmenes de información disponible que no presentan un nivel de estructuración tan fuerte y que, sin embargo, sería conveniente poder utilizar.

Además, es importante notar que, incluso cuando los repositorios o fuentes base para un sistema de integración son todos estructurados, sigue habiendo importantes dificultades a abordar, tales como las heterogeneidades de esquema, de las taxonomías utilizadas, de los formatos de representación de datos, etc.

Por ello, existe hoy en día destaca un fuerte interés dedicado a los temas relacionados con su integración, centrado fundamentalmente en el área de Bases de Datos Federadas[BKLW99][IBM98][Bla96][FRV96][HZ96][HKWY97][TRV98]

II.2.3. FUENTES SEMIESTRUCTURADAS

En un número muy importante de fuentes, cuyo máximo exponente pueden ser las fuentes web, ocurre que los datos presentan una cierta estructura pero que ésta es irregular. En general, de hecho, en un sitio web no habrá un esquema explícito para los datos.

Por ello, parece una buena idea disponer de algún tipo de modelo de representación de datos que permita modelar fuentes de este tipo. Los modelos de datos

semiestructurados fueron introducidos en trabajos como [PGMW95] y sus características son aplicables a los problemas antes mencionados.

Habitualmente entendemos por datos semiestructurados a aquellos que presentan alguna de las siguientes características[FLM98] :

- El esquema de los datos no es conocido de antemano. Puede existir, pero estará implícito en los datos y habrá que inferirlo de alguna manera.
- El esquema de los datos es relativamente grande y puede cambiar con frecuencia.
- El esquema describe el estado actual de los datos, pero tolera violaciones del esquema en ciertos casos, que no siempre son especificados.
- No hay un tipado de datos fuerte. En ocasiones, para un mismo atributo podemos encontrar valores de tipos diferentes.

Los modelos propuestos para representar datos semiestructurados han estado generalmente basados en grafos dirigidos y etiquetados[Abi97].

En este caso los nodos de los grafos representan objetos de datos. Cada arco representa un atributo de dicho objeto cuyo valor es la etiqueta del arco. Es importante notar que, hablando de datos semiestructurados, y de acuerdo a los puntos antes mencionados, a menudo no es posible hacer aserciones estrictas sobre el número de arcos que saldrán de un determinado nodo ni sobre el tipo de datos de cada atributo.

Un buen ejemplo de información *semiestructurada* puede ser la página de resultados de una búsqueda de libros realizada sobre una tienda electrónica en Internet. El resultado obtenido es una página HTML con una serie de información adicional en la que aparece insertada la lista de libros obtenidos. La información de cada uno de los libros se presenta con un formato similar (como se puede observar en la Figura II.2.3.1) que es el que, implícitamente, define un cierto esquema para los datos.

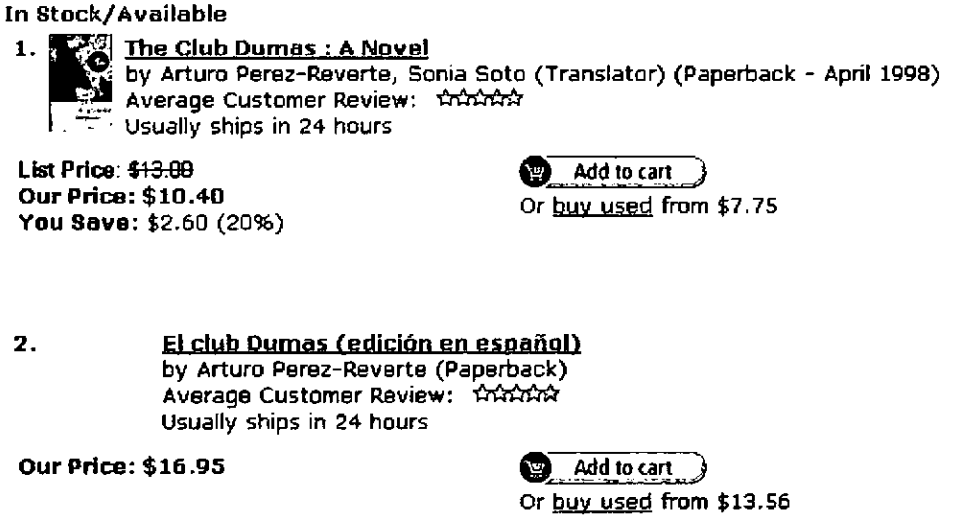


Figura II.2.3.1 Resultado típico de una consulta a una tienda web

Sin embargo, se pueden apreciar heterogeneidades en la representación de los items (como por ejemplo que para algunos libros aparece un descuento asociado ó una fotografía de la portada del libro y en otros casos no). Es posible además que cierta información adicional se pueda obtener accediendo a otra página atravesando el enlace sobre el título del libro, obligando a una navegación extra.

Realizar consultas estructuradas sobre este tipo de fuentes es posible pero, normalmente, involucra vencer una serie de importantes dificultades, además de las derivadas de la escasa rigidez de su esquema:

- Capacidades de consulta limitadas: A menudo ocurre que estas fuentes no permiten ser consultadas arbitrariamente, sino que exponen algún tipo de interfaz más limitado al que deben ceñirse las consultas realizadas. Por ejemplo en una fuente web como la del ejemplo anterior, las consultas posibles están limitadas por un formulario de búsqueda que puede permitir la consulta sólo por ciertos atributos (e.g. título, autor).
- Acceder a los datos de respuesta de una consulta puede involucrar la realización automática de algún proceso de acceso a la fuente y de navegación por sus páginas. Por ejemplo para implementar una consulta sobre una fuente web, puede haber que establecer una sesión en la misma y rellenar automáticamente formularios de consulta mediante alguna secuencia de peticiones HTTP[HTT97].
- Extraer los datos. Frecuentemente la respuesta obtenida a una consulta sobre este tipo de fuentes, no será devuelta en un formato que identifique claramente las tuplas obtenidas como respuesta y los atributos constituyentes de las mismas. Por ejemplo, en una fuente web, dichas tuplas vendrán embebidas en una página HTML[HTM99] sin que el formato de la misma permita identificarlas directamente. Habrá que realizar, pues, algún proceso de análisis de la página para poder extraer automáticamente las tuplas.

Por lo tanto, tal y como puede verse, existen fuertes dificultades a vencer para poder tratar este tipo de fuentes como estructuradas y así poder proceder a su integración. Sin embargo, debido a la gran cantidad de información semiestructurada existente en la actualidad (estimaciones recientes afirman que aproximadamente el 90% de la información disponible está almacenada fuera de una Base de Datos Relacional), sin duda el esfuerzo puede valer la pena.

Algunos ejemplos comunes de fuentes semiestructuradas son los siguientes:

- Cualquier base de datos accesible en una fuente web a través de un típico formulario de consulta, como por ejemplo el catálogo de una tienda en línea o la base de datos de ofertas de empleo de un sitio de empleo en línea.
- Cualquier documento en cualquier formato (e.g. PDF, Microsoft Word) que presente un listado más o menos homogéneo de items, como por ejemplo el catálogo impreso de los productos de una determinada empresa.
- Cualquier colección de documentos XML que presenten marcas opcionales, heterogeneidad de taxonomías o esquema, etc.

II.3. CLASIFICACIÓN DE LOS SISTEMAS DE INTEGRACIÓN DE INFORMACIÓN DISTRIBUIDA E HETEROGÉNEA

Existen diversos enfoques que pueden adoptarse para abordar el problema de integrar fuentes distribuidas y heterogéneas. En este apartado se repasan someramente.

Una clasificación de los sistemas de integración de datos procedentes de fuentes heterogéneas puede verse en [DD99]. La clasificación presentada en este apartado utiliza conceptos y nomenclaturas procedentes de dicho trabajo, si bien se han introducido algunas modificaciones significativas.

En primer lugar, es conveniente fijar el alcance de esta clasificación: se pretende caracterizar a aquellos sistemas capaces de acceder a los datos de fuentes de consulta heterogéneas de una manera coherente, unificada y aportando un valor añadido con respecto a la consulta directa de las mismas. También es importante destacar que el foco de la clasificación está en la consulta y no en la manipulación de datos. Lo que esto quiere decir es que la manipulación de información se produce fundamentalmente en las fuentes, quedando el papel fundamental del sistema integrador en unificar e integrar la consulta sobre las mismas.

Utilizaremos dos criterios fundamentales para realizar la clasificación.

El primero de ellos distingue entre los sistemas de integración de datos que utilizan el enfoque *virtual* y aquellos que utilizan el enfoque *materializado*. El apartado II.3.1 se ocupa de esta distinción.

El segundo criterio de clasificación divide los sistemas de integración de datos en función del tipo de fuentes que tratan y del tipo de consultas que son capaces de responder. Pueden distinguirse tres tipos fundamentales de sistemas de acuerdo a este criterio:

- Motores de indexación y metabúsqueda: Son sistemas que tratan fuentes no estructuradas permitiendo consultas imprecisas sobre las mismas.
- Sistemas de integración de Bases de Datos: Sistemas que tratan fuentes estructuradas permitiendo consultas precisas.
- Mediadores: Sistemas que tratan fuentes estructuradas y semiestructuradas permitiendo consultas precisas. Opcionalmente pueden permitir consultas imprecisas mediante la integración de algún sistema basado en las técnicas de motores de indexación.

Los apartados II.3.2, II.3.3, II.3.4 tratan respectivamente cada una de estas clases de sistemas.

II.3.1. SISTEMAS VIRTUALES Y SISTEMAS MATERIALIZADOS

Existen dos aproximaciones bien diferenciadas a la hora de construir sistemas de integración de datos distribuidos:

- Sistemas materializados. En este caso, los datos procedentes de las múltiples fuentes son importados periódicamente y volcados en un *almacén de datos*. Todas las consultas pueden entonces realizarse sobre ese almacén.
- Sistemas virtuales. En esta aproximación, los datos permanecen en las fuentes y las consultas al sistema de integración se descomponen en tiempo de ejecución, en subconsultas adecuadas para cada fuente. El sistema de integración combina e integra las respuestas proporcionadas por cada fuente para devolver una respuesta unificada a la consulta general.

El primer enfoque tiene la desventaja de que es necesario actualizar el *almacén* cuando los datos cambian. Como esto no siempre es posible o fácil de realizar, a menudo los datos se encuentran sin actualizar.

Además, no siempre será posible copiar todos los datos del sistema original en el almacén. Por ejemplo, en el caso de fuentes web, no siempre la interfaz disponible para consultas permite extraer *todos* los datos. Por ejemplo, una tienda virtual de libros normalmente permitirá buscar en su catálogo por título o autor, pero no permitirá acceder a una lista de todos los libros de su catálogo, de manera que éste pueda ser almacenado completo en el sistema de integración. Otra razón por la que puede no ser posible copiar todos los datos es que, sencillamente, su volumen puede ser demasiado grande para ser transferido por la red en un tiempo razonable.

En el lado positivo, este enfoque puede garantizar un rendimiento adecuado en el momento de realizar las consultas, ya que todos los datos residen localmente al sistema de integración. Es apropiado cuando estamos ante datos que varían muy lentamente o para procesos en que no es necesario que los datos estén plenamente actualizados (como, por ejemplo, para ejecutar procesos estadísticos o de minería de datos).

En la aproximación virtual, los datos no están replicados, y por tanto se garantiza su actualidad en el momento de realizar la consulta, pero por contra, debido a la autonomía de las fuentes, se necesitan métodos de optimización y ejecución de consultas más sofisticados para garantizar un rendimiento adecuado. Para tratar de minimizar los problemas derivados de esto, la mayoría de los sistemas virtuales utilizan algún tipo de sistema de cache que almacena localmente parte de los datos de las fuentes.

En general, es aceptado que la aproximación virtual es más apropiada para construir sistemas en los que el número de fuentes de información es amplio, los datos cambian frecuentemente, y hay un control pequeño sobre las fuentes (también se utiliza en aquellos casos en los que la información externa no puede ser almacenada en un *almacén*, como, por ejemplo, si no lo permiten las capacidades de consulta de la fuente

original). Por ello, es el enfoque más utilizado en los modernos sistemas de integración de datos.

II.3.2. MOTORES DE INDEXACIÓN Y METABÚSQUEDA

Si bien el estudio en profundidad de este tipo de sistemas no se encuentra entre los objetivos de este trabajo, se proporciona en este epígrafe una breve descripción de los mismos.

Tal y como ya se ha comentado, los motores de indexación basan su funcionamiento en la construcción de un índice de palabras sobre una colección de documentos y, en permitir, posteriormente, consultar dicho índice mediante una consulta imprecisa.

Este tema ha sido tratado de manera muy extensa por la comunidad que estudia las llamadas técnicas de Recuperación de Información[Bae98]. Algunos de los principales problemas que es necesario abordar en la construcción de este tipo de sistemas son la minimización del tamaño de los índices generados y la consulta eficiente sobre los mismos, la ordenación por relevancia de los resultados obtenidos en una consulta y la construcción de arquitecturas distribuidas capaces de manejar eficientemente grandes volúmenes de consultas.

Basándose en estas técnicas se han construido sistemas capaces de indexar de manera eficiente grandes colecciones de documentos, como en el caso del WWW. Así, cualquiera de los buscadores de Internet convencionales[Alt][Lyc], basa su funcionamiento en la construcción de un gigantesco índice de palabras que constituye una abstracción no estructurada de la enorme colección de documentos de la que consta el WWW estático, permitiendo así consultas imprecisas sobre el mismo.

En este ámbito específico de la consulta imprecisa del WWW, en los últimos años han aparecido algunos trabajos que sacan provecho de su estructura basada en hiperenlaces para mejorar la eficiencia de indexación y consulta, así como la eficacia de los algoritmos de ordenación por relevancia. Sin duda, el trabajo más conocido en este ámbito es el realizado dentro del buscador Google[BP98]. Otros trabajos que utilizan la información estructural del WWW son [PPR96][CK98].

Otra línea de investigación ha apostado por utilizar métodos estadísticos para encontrar similitudes entre documentos e inferir relaciones temáticas entre los mismos. Estas técnicas permiten realizar tareas como la clasificación automática de documentos en categorías temáticas o el descubrimiento de documentos relevantes de un tema determinado. Estas tecnologías son la base de numerosos productos comerciales (como Autonomy[Aut]) que se han utilizado fundamentalmente para propósitos de gestión del conocimiento y gestión documental dentro de grandes organizaciones.

La manera de consultar las colecciones de documentos indexadas con este tipo de motores sigue siendo la misma que en los casos anteriores, con el único añadido de que

las búsquedas pueden restringirse a los documentos de una temática determinada y es el mismo motor el que trata de determinar, normalmente a través de un conjunto de documentos de ejemplo, qué documentos pertenecen a la temática solicitada.

Otro tipo de sistemas que permiten consultas imprecisas son los llamados metabuscadores. Un metabuscador proporciona un punto de entrada único para la realización de una misma consulta sobre varios motores de indexación a la vez. El metabuscador, al recibir una consulta, decide qué motores de búsqueda de entre los disponibles en el sistema pueden proporcionar una respuesta a la misma, y la emite simultáneamente contra todos ellos. Al recibir las respuestas, las coteja e integra para devolver una respuesta unificada a la consulta recibida. Por lo tanto, los metabuscadores utilizan el enfoque virtual.

Los metabuscadores han sido utilizados, especialmente en el ámbito de la búsqueda en el WWW, para ampliar la extensión de las colecciones de documentos sobre las que se emiten consultas y para sacar provecho de la respuesta de los diferentes motores de búsqueda para conseguir una mejor ordenación por relevancia de los resultados. Algunos sistemas relevantes son MetaCrawler[Met], SavvySearch[DH97] y, en el ámbito español, el Multibuscador de Biwe [Mul].

Por último, es interesante destacar que una de las maneras de construir metabuscadores es mediante el uso de un mediador sobre los motores de búsqueda en los cuales se metabusca.

II.3.3. SISTEMAS DE INTEGRACIÓN DE BASES DE DATOS

Para integrar los datos de diversas bases de datos heterogéneas, podemos distinguir tres enfoques fundamentales: Base de Datos Universal, Almacén de Datos y Bases de Datos Federadas. Los siguientes epígrafes se ocupan, respectivamente, de cada uno de ellos.

II.3.3.1. BASE DE DATOS UNIVERSAL

Consiste en la creación de una nueva base de datos cuyo esquema unifique el de todas las bases de datos a integrar. Las antiguas bases de datos dejan de funcionar y todas las operaciones pasan a realizarse contra la nueva, incluidas las de escritura.

Este enfoque tiene el grave inconveniente de que todas las aplicaciones que estaban funcionando sobre las viejas bases de datos, dejan de funcionar y, por lo tanto, es necesario reprogramarlas o desarrollar otras nuevas. Como consecuencia, posiblemente también sea necesario realizar programas de formación en el uso de las nuevas aplicaciones, para los usuarios de las viejas.

Además, será necesario hardware nuevo más potente que el anterior para que el sistema sea capaz de realizar todas las operaciones que antes realizaba por separado cada una de las bases de datos. Por lo tanto, las migraciones son largas y muy caras.

Otro inconveniente importante es que si en el futuro es necesario integrar otra base de datos, todo el sistema puede verse afectado, lo que compromete gravemente la extensibilidad del enfoque.

Como ventaja, este enfoque será más eficiente que el resto, ya que, al fin y al cabo, una vez realizado, no deja de ser una base de datos convencional.

II.3.3.2. ALMACÉN DE DATOS ('DATA WAREHOUSE')

Al igual que en el enfoque anterior, se construye una nueva base de datos unificada que contiene datos de todas las bases a integrar.

La diferencia fundamental con el enfoque anterior es que, en este caso, las bases de datos originales no desaparecen sino que siguen funcionando. Los datos del almacén se utilizan sólo para propósitos de lectura (normalmente para realizar algún proceso de análisis o minería de datos sobre los mismos), mientras que el resto de procesos siguen funcionando de la misma manera que antes de la creación del almacén.

El proceso de importación de los contenidos desde las bases de datos a integrar al almacén de datos se realiza periódicamente. Durante el proceso de importación, los datos pueden ser transformados de alguna manera para ajustarse mejor a los procesos de análisis que van a conducirse sobre ellos en el almacén.

Como las bases de datos originales siguen existiendo, las viejas aplicaciones siguen funcionando y, por lo tanto, no es necesario rehacerlas ni volver a formar a sus usuarios. La inversión adicional en hardware sigue siendo necesaria para poder alojar el almacén de datos, pero frecuentemente los requerimientos serán menores, ya que el sistema sólo tendrá que responder a las necesidades de un número de procesos restringido.

Un importante inconveniente de este enfoque es su carácter estático: los datos del almacén no están actualizados, por lo que sólo pueden realizarse sobre ellos consultas en las que el ratio de actualización no sea crucial. Además, como el volumen de datos es normalmente muy grande, las actualizaciones no pueden hacerse fácilmente con periodicidades cortas. De hecho, en la mayoría de ocasiones, el almacén no contiene todos los datos de las bases originales, sino sólo un subconjunto de los mismos. Esta es la razón por la que los almacenes de datos se utilizan primordialmente para tareas de análisis y minería de datos sobre históricos y no para otro tipo de labores.

II.3.3.3. BASES DE DATOS FEDERADAS

El propósito de los sistemas de bases de datos federadas es integrar diversas bases de datos dispersas y heterogéneas para proporcionarle al usuario la impresión de estar consultando una única base de datos unificada.

A diferencia de los otros dos sistemas de integración de fuentes estructuradas, los sistemas de bases de datos federadas no utilizan el enfoque materializado, sino el virtual.

Habitualmente, el administrador del sistema federado puede contemplar las tablas de las bases de datos remotas como si estuviesen disponibles localmente. Entonces puede definir las relaciones del esquema global que exportará el sistema federado, mediante la definición de vistas sobre las tablas de las bases originales, de manera similar a como definiría una vista local en una base de datos convencional.

El sistema federado es capaz entonces de admitir consultas expresadas sobre las relaciones del esquema global y escritas en algún lenguaje estructurado como SQL[SQL92] u OQL[OQL].

Cuando el sistema federado recibe una consulta, la subdivide en subconsultas para cada una de las bases de datos individuales. Cada base de datos resuelve su subconsulta enviando el resultado al sistema integrador. Este recupera los resultados proporcionados por cada una de las fuentes y los integra para devolver un resultado a la consulta general conforme a las relaciones del esquema global.

Entre los principales problemas a los que tiene que enfrentarse un sistema de bases de datos federadas se encuentran los siguientes:

- Reformulación y ejecución de consultas. Gran parte de los mecanismos convencionales para reformular consultas en vistas son aplicables. Sin embargo, las consultas emitidas a las fuentes deben reescribirse para ser hechas en el lenguaje nativo de la base de datos remota. Esto puede ser especialmente complejo si las bases de datos siguen modelos diferentes (e.g Relacional vs. Orientado a Objetos). El motor de ejecución de consultas debe también tener en cuenta aspectos como el acceso en paralelo a las diversas fuentes o la integración de los resultados obtenidos.
- Optimización de consultas. El optimizador debe ser capaz de trabajar con información de costes heterogénea y que quizás sea incompleta. Además debe tener en cuenta aspectos como los retardos de red o la diversidad de plataformas de ejecución de las bases de datos remotas.
- Diversidad de representación. Un mismo objeto del mundo real puede, sin embargo, tener una representación diferente en las bases de datos remotas. Sin embargo, el sistema federado debe ser capaz de reconocer a ese objeto en sus diferentes representaciones para poder realizar correctamente diversas operaciones,

como por ejemplo un join. Este es un problema difícil de tratar que suele abordarse con heurísticas específicas del dominio de aplicación.

Los sistemas de Bases de Datos Federadas han tenido una larga trayectoria en los laboratorios de investigación[BKLW99][Bla96][FRV96][HZ96][HKWY97][TRV98]. Más recientemente, han aparecido sistemas comerciales que proporcionan parte de las funcionalidades requeridas para realizar estas tareas, como DataJoiner[IBM98] de IBM.

Este enfoque presenta importantes ventajas sobre los dos previos:

- No es necesario realizar ningún proceso de reingeniería de las aplicaciones utilizadas sobre las bases de datos originales (ventaja sobre el enfoque de 'Base de Datos Universal').
- El nuevo hardware necesario es mínimo, ya que el sistema de bases de datos federadas sigue delegando el grueso del trabajo en las bases originales, limitándose a las tareas de integración (ventaja sobre los dos enfoques previos).
- Los datos están permanentemente actualizados y están todos disponibles, con lo que pueden utilizarse para cualquier tipo de aplicación y no sólo aquellas relacionadas con la minería de datos sobre históricos (ventaja sobre el enfoque de 'Almacén de Datos'). Si el administrador lo desea, en algunos casos le será permitido mantener caches de los datos de las bases originales, pero eso será siempre una decisión que podrá tomar en función de la naturaleza de la aplicación a construir.

El inconveniente que presenta este enfoque es que, al residir los datos en las bases originales, los accesos a través del sistema federado pueden ser menos eficientes. Para tratar de paliar en lo posible este problema, la mayoría de sistemas de bases de datos federadas se apoyan en algún tipo de cache o sistema de precarga de datos.

II.3.4. SISTEMAS BASADOS EN MEDIADORES

Los sistemas basados en mediadores son en cierta manera similares a los de bases de datos federadas, con la diferencia de que su ámbito de actuación se extiende más allá de las fuentes estructuradas.

Un sistema mediador tratará, al menos, la integración de fuentes de datos estructuradas y semi-estructuradas, sacando provecho de la estructura parcial de estas últimas para permitir, también sobre ellas, la realización de consultas precisas escritas en lenguajes de consulta estructurados como los utilizados sobre bases de datos convencionales (y, por lo tanto, estructuradas).

Además, los mediadores pueden incluir algún tipo de tecnología de motor de indexación para proporcionar ciertas capacidades para la realización de consultas imprecisas sobre los datos. Esto puede ser útil para ciertas aplicaciones que no

requieran un alto grado de precisión en las consultas pero deseen ofrecer interfaces de búsqueda simples. La funcionalidad más común en este caso consiste en que el mediador permite localizar todas aquellas tuplas de una relación que satisfacen una determinada consulta por palabra clave en alguno de sus campos, sin que el usuario tenga que especificar explícitamente en cuál de ellos.

Al igual que en el caso de los sistemas de bases de datos federadas, el enfoque utilizado en los mediadores, es el virtual (a veces auxiliado por un sistema de cache).

También al igual que en el caso de los sistemas de bases de datos federadas, el sistema mediador mantiene un esquema mediado sobre el que los usuarios emiten las consultas. A continuación, se define este concepto con mayor detalle.

Un esquema mediado (o *esquema global* o *esquema de mediación*) es un conjunto de relaciones virtuales que han sido diseñadas para una aplicación concreta de integración de datos. Estas relaciones "virtuales" se comportan, desde el punto de vista del usuario, de manera muy similar a las relaciones de una base de datos convencional (e.g. las tablas de una base de datos relacional). Sin embargo, tal y como corresponde al enfoque virtual, las tuplas que componen las relaciones del esquema mediado no están realmente almacenadas en ningún sitio (si exceptuamos posibles sistemas de cache), sino que es el mediador quién las compone dinámicamente partiendo de los datos de las fuentes cuando una consulta del usuario así lo requiere.

Así, las consultas emitidas por los usuarios del mediador se referirán a las relaciones del esquema mediado y las respuestas devueltas deben ser, por lo tanto, también conformes a ellas.

Como consecuencia, al recibir una consulta del usuario, la primera tarea para el mediador es reformularla como un conjunto de subconsultas que se refieran directamente a los esquemas de las fuentes de información.

Para ejecutar estas subconsultas, cada fuente tendrá asignado un *programa envoltorio* (o *wrapper*) que será el encargado de resolver las subconsultas enviadas sobre ella. Esto involucra que el envoltorio de una fuente debe ser capaz de:

1. Traducir las subconsultas del lenguaje de consulta general al lenguaje de consulta nativo de la fuente. Este paso puede ser complejo ya que, por ejemplo en una fuente web, este "lenguaje nativo" consistirá en la invocación a través de HTTP[HTT97] de algún tipo de formulario de consulta.
2. Traducir el formato de salida de resultados de la fuente al formato general. Nuevamente esta tarea puede ser significativamente compleja. Siguiendo con el ejemplo de las fuentes web, su formato de salida típico es HTML, que debe ser traducido a un modelo más estructurado como el relacional.

Posteriormente, con los resultados obtenidos de cada fuente, el sistema mediador debe ser capaz de combinarlos e integrarlos para devolver así al usuario una respuesta unificada a la consulta global.

Ejemplo II.3.4.1: Un esquema mediado simple

Supóngase que se dispone de dos fuentes de información A, B que representan a dos sitios web que proporcionan información sobre empresas. A proporciona información sobre sus cotizaciones bursátiles mientras B proporciona información general obtenida del registro mercantil, como, por ejemplo, la identidad de su accionista principal. Supóngase que A y B son modeladas como dos relaciones con los siguientes atributos:

```
A={NOMBRE_EMPRESA, CIF, SÍMBOLO_BURSATIL, COTIZACION}
B={NOMBRE_EMPRESA, CIF, ACCIONISTA_PRINCIPAL}
```

Un mediador permitiría la definición de una relación R en el esquema mediado que combine la información de A y B para ofrecer tuplas con los siguientes atributos:

```
R={NOMBRE_EMPRESA, CIF, SÍMBOLO_BURSÁTIL, COTIZACIÓN,
ACCIONISTA_PRINCIPAL}
```

Supóngase ahora que el mediador recibe la siguiente consulta del usuario sobre R:

```
Select * from R where NOMBRE_EMPRESA='Acme'
```

Para responder adecuadamente a la consulta, el mediador realizará las siguientes tareas:

1. Reformular la consulta recibida sobre R en las siguientes dos subconsultas sobre A y B:

   ```
   Select * from A where NOMBRE_EMPRESA='Acme'
   Select * from B where NOMBRE_EMPRESA='Acme'
   ```

2. Ejecutar cada una de las subconsultas sobre las fuentes a través de sus programas envoltorio, obteniendo los resultados para las mismas. En este caso, al tratarse de fuentes web, los envoltorios tendrán que ser capaces de:
   i. Emular las subconsultas SQL recibidas utilizando una o varias invocaciones mediante HTTP de formularios de consulta disponibles en la fuente web.
   ii. Analizar las páginas HTML recibidas como respuesta para extraer de ellas las tuplas que constituyen la respuesta a la subconsulta.
3. Fusionar los resultados obtenidos de cada subconsulta haciendo el join entre ellos (por ejemplo usando el atributo CIF).
4. El resultado del join es el resultado a la consulta emitida por el usuario

□

Analizando el ejemplo anterior queda de manifiesto que en la construcción de un sistema mediador es necesario abordar todos los problemas con los que también se encuentran las bases de datos federadas. Sin embargo, la integración de fuentes semiestructuradas y con capacidades de consulta limitadas, plantea un conjunto de importantes problemas adicionales:

- Capacidades de consulta limitadas: A diferencia de las bases de datos, muchas fuentes semiestructuradas, como las fuentes web, no permiten la realización de cualquier consulta sobre sus relaciones, sino que están limitadas a las permitidas por algún tipo de interfaz de acceso (e.g. un formulario de consulta en una fuente web). Estas capacidades de consulta deben ser representadas. Además, deben calcularse las capacidades de consulta de la relaciones del esquema mediado partiendo de las capacidades de las fuentes.
- El acceso y recuperación de los datos es a menudo complejo. Ya se ha comentado que, por ejemplo, realizar una consulta sobre una fuente web puede requerir establecer una sesión en ella, rellenar un formulario de consulta y analizar la página HTML obtenida como respuesta para extraer de ella las tuplas de respuesta. Todos estos problemas deben ser abordados por el programa envoltorio de la fuente. Un sistema mediador debe proporcionar mecanismos que ayuden en la generación de estos programas, al menos para los tipos de fuentes más comunes.
- La naturaleza semiestructurada de los datos a tratar requiere que el mediador permita mayor flexibilidad en la manera de consultarlos, combinarlos e integrarlos. Si bien el mediador debe tratar de ofrecer una visión de los datos lo más estructurada posible, debe proporcionar también mecanismos para tratar estas peculiaridades: por ejemplo puede utilizar un sistema de tipado menos estricto o proporcionar herramientas para manejar heterogeneidades en los formatos de representación utilizados por una fuente para un determinado atributo.
- Debido a la abundancia de fuentes semiestructuradas, es frecuente que los sistemas mediadores traten con un número de fuentes mucho mayor que los sistemas de bases de datos federadas. Además, normalmente, las fuentes presentan un mayor grado de autonomía y existen pocos metadatos sobre ellas, por lo que el sistema debe ser capaz de funcionar en ausencia de este tipo de informaciones (e.g. información relativa a costes).

II.4. MOTIVACIÓN

Tal y como ya se ha comentado previamente, si bien el alcance de su aplicabilidad es mayor, los modernos sistemas de integración de información han surgido, en gran medida, bajo el impulso de dos ámbitos concretos de aplicación, que han levantado importantes expectativas industriales: aplicaciones de búsqueda/agregación en Internet por un lado, y aplicaciones de apoyo al negocio electrónico y la gestión del conocimiento dentro de organizaciones medianas o grandes, por otro. Los siguientes subapartados aportan algo más de detalle sobre ambos ámbitos.

II.4.1. APLICACIONES INTERNET DE BÚSQUEDA / AGREGACIÓN / COMPARACIÓN

En los últimos años han aparecido multitud de sitios web que, normalmente con propósitos comerciales, proporcionan algún tipo de servicio a los usuarios de Internet.

Algunos ejemplos populares son: las tiendas electrónicas que venden todo tipo de productos, sitios web de ofertas/demandas de empleo, servicios bancarios y financieros (tales como las bancas electrónicas), medios de comunicación, centros de subastas y un amplísimo etecétera.

Sin embargo, casi siempre estos servicios están distribuidos en diferentes sitios web y no resulta sencillo combinarlos o unificarlos en aras de compararlos, obtener una mayor potencia o una superior comodidad de uso.

Hay tres tipos fundamentales de aplicaciones que surgen en este contexto y que pueden beneficiarse enormemente del uso de nuevas técnicas de integración de datos: aplicaciones de comparación, de búsqueda y de agregación.

Los siguientes subapartados proporcionan ejemplos de este tipo de aplicaciones con algo más de detalle. Posteriormente, se exponen algunas consideraciones generales sobre la manera en que los sistemas mediadores pueden facilitar su construcción.

II.4.1.1. APLICACIONES DE COMPARACIÓN

En muchas ocasiones, un mismo producto o servicio es ofrecido por una larga lista de proveedores, de manera que para el usuario es un proceso largo y tedioso, cuando no imposible, determinar qué proveedor le ofrece las condiciones que mejor se ajustan a sus necesidades.

Tomemos el ejemplo de la compra de productos. Un usuario que desee realizar la compra de un determinado producto (e.g. un libro) se encontrará con una amplia oferta de tiendas electrónicas que tienen dicho producto a la venta. Sin embargo, la única manera de que dispone dicho usuario para encontrar aquella tienda que le ofrece

mejores condiciones, requiere que se conecte una a una a las tiendas (que pueden ser cientos), y compruebe las condiciones en cada una, anotándolas para su posterior comparación.

Claramente sería deseable para el usuario disponer de una manera de obtener en un solo paso las mejores condiciones para un producto dado. Esta tarea puede ser abordada desde el enfoque de integración de información. Si se integra la información de las diferentes tiendas electrónicas, sería posible para el usuario obtener con una simple consulta sobre el repositorio unificado, aquella tienda que le ofrece mejores posibilidades.

II.4.1.2. APLICACIONES DE BÚSQUEDA

Frecuentemente, diversas fuentes web ofrecen información, productos o servicios de un determinado dominio de aplicación. Un usuario que desee localizar algún item concreto en ese dominio, precisaría visitar una a una las posibles fuentes y efectuar una búsqueda en ellas.

Por ejemplo, supóngase un usuario de Internet que esté buscando ofertas de trabajo con unas determinadas características (e.g. ofertas en la categoría de 'Finanzas', para trabajar en 'La Coruña' o 'Madrid', y con un sueldo superior a una cierta cantidad de dinero). Para encontrar todas las que podrían interesarle, precisaría recorrer todos los sitios web de empleo, realizar una búsqueda en ellos, especificando en sus formularios de consulta las condiciones indicadas previamente, y anotar todas las ofertas obtenidas. Claramente sería deseable que el usuario pudiese obtener todas las ofertas realizando una única consulta.

Las aplicaciones de búsqueda más populares en Internet en la actualidad, se basan en la idea de *araña* de búsqueda. Una araña de búsqueda, como pueden ser Altavista[Alt], Lycos[Lyc] o Google[BP98], utiliza los hiperenlaces de los sitios web para "descubrir" páginas HTML y construye un índice por palabra de la información contenida en ellas. Una vez construido el índice, permite realizar búsquedas por palabra clave sobre él.

Este enfoque, aunque atractivo para muchos tipos de búsqueda, es inválido para el caso que nos ocupa ya que :

- La visión que proporcionan de la información de la web es no estructurada. Como consecuencia de esto, sólo permiten la realización de consultas por palabra clave y no permiten ningún tipo de consulta estructurada. Por lo tanto, una consulta como la planteada en nuestro ejemplo, no puede ser realizada.
- Presentan un nivel de granularidad de página. Esto quiere decir que los resultados de una búsqueda son localizadores de páginas que pueden tener información relevante a la consulta. No son realmente los items buscados.

- No permiten consultar el llamado *web dinámico* o *web oculto*. Con estos términos se hace referencia a las páginas web que sólo son accesibles mediante el uso de un formulario de búsqueda HTML. Estimaciones recientes consideran que el web dinámico constituye más del 90% de la información contenida en el total de la web. Los buscadores tipo araña no son capaces de acceder a esas páginas y, por lo tanto, no pueden indexar sus contenidos. La información relevante en nuestro ejemplo es justamente de este tipo.

Las mismas consideraciones que se han realizado para el caso de la búsqueda de ofertas de empleo, son extrapolables para la búsqueda de cualquier elemento de información que tenga una cierta estructura y sea accesible sólo a través de consultas mediante formularios web.

II.4.1.3. APLICACIONES DE AGREGACIÓN DE CUENTAS

A menudo, un usuario de Internet tiene diversas cuentas abiertas con diferentes proveedores de algún tipo de servicio. Por ejemplo, un usuario puede tener abiertas diversas cuentas de correo electrónico web. Otro ejemplo puede ser el caso de un usuario que tiene abiertas cuentas en las diversas bancas electrónicas de los bancos con los que trabaja.

De esta manera, si un usuario desea saber el saldo total de todas sus cuentas bancarias o leer el nuevo correo que tenga en todas sus cuentas, tendrá que ir recorriendo uno a uno los diferentes sitios web.

Nuevamente, sería deseable que pudiese disponer de un único frontal sobre el que pudiese realizar dichas operaciones de manera agregada.

□

Una vez descritos los principales tipos de aplicaciones Internet en los que es necesario introducir nuevos sistemas de integración de datos, las siguientes líneas exponen algunas consideraciones generales sobre la manera de abordar dicho problema.

Para todas estas aplicaciones, un enfoque de unificación de fuentes web basado en algún tipo de almacén de datos único (o *data warehouse*. Ver apartado II.3.3.2) al que se volcasen los datos de todas las fuentes (el enfoque tradicional en otro tipo de aplicaciones), es inviable por una serie de razones, algunas de las cuales son: 1) Frecuentemente el número de fuentes es tan grande que el volumen de datos sería muy complejo de manejar, cuando no imposible 2) Debido a su volumen, los datos no podrían ser actualizados frecuentemente (de hecho, por ejemplo en una tienda electrónica, el volumen puede ser tan grande que la misma idea de transferirlos por la red sea absurda) 3) Añadir nuevas fuentes al sistema sería complejo y costoso. La razón es que no es posible aprovechar la infraestructura ya existente de la misma. Un nuevo desarrollo debe ser realizado de manera *ad-hoc* para transformar los datos de la fuente al formato del almacén. 4) Dada la autonomía de las fuentes, este enfoque puede

ser sencillamente imposible, si estas no proporcionan sus datos y no ofrecen capacidades de consulta suficientes para obtenerlos en su totalidad directamente desde su sitio web.

Un enfoque que ofrece una flexibilidad muy superior para la integración de información de fuentes Internet, es el de mediador. Las fuentes web comentadas son un buen ejemplo de fuente de datos semi-estructurados. Por ejemplo, los listados de productos que ofrece una tienda electrónica al navegar por sus páginas o realizar una búsqueda contra su catálogo, siguen una cierta estructura, ya que están compuestos por una serie de campos más o menos fijos que se repiten para cada uno de los productos. Sin embargo, esta estructura no atiende un esquema definido explícitamente (éste debe deducirse de los datos) y admite variaciones (por ejemplo, algunos atributos pueden no estar disponibles para todos los productos. El formato de representación de un atributo también puede variar de un producto a otro).

Otras importantes dificultades adicionales también características de las fuentes web son que: 1) Las fuentes suelen tener limitadas sus capacidades de consulta. De hecho, normalmente sólo podrán realizarse aquellas permitidas por un determinado formulario de consulta. 2) La realización de consultas involucra algún tipo de navegación a través de la fuente usando el protocolo HTTP 3) Los resultados a las consultas se encuentran embebidos en páginas HTML, donde no hay información que permita identificar los resultados obtenidos a una consulta ni su división en atributos.

Si bien estas dificultades son complejas de vencer, si se consigue, los beneficios son grandes, ya que: 1) no es necesario disponer de un almacén de datos central 2) los datos están permanentemente actualizados ya que se consultan directamente en las fuentes (si bien es posible mantener una cache refrescada a pequeños intervalos) 3) Asumiendo que el mediador dispone de mecanismos adecuados para la generación de programas envoltorio, añadir nuevas fuentes es rápido, poco costoso y no requiere de trabajo adicional por parte de la fuente de datos.

Por último, decir que las aplicaciones en ámbito Internet de búsqueda, agregación y comparación, se caracterizan por: 1) Un número alto de fuentes (que pueden ser varias centenas) 2) Un muy alto grado de autonomía de las mismas. En muchos casos, la autonomía puede ser total.

Por ello, los mecanismos para la generación y mantenimiento de programas envoltorio son fundamentales.

II.4.2. APLICACIONES DE INTEGRACIÓN DE INFORMACIÓN EN ENTORNOS CORPORATIVOS

Los cambios que se están produciendo en el entorno de negocio actual están llevando a que las organizaciones modernas se encuentren con que su funcionamiento normal genera una cantidad de información enorme, que es difícil de gestionar y de la que es muy complicado extraer valor.

Además, una multitud de factores tales como la rápida evolución tecnológica, las presiones del mercado o el efecto de la competencia entre diferentes proveedores de soluciones técnicas, ha causado que los entornos y tecnologías utilizadas para la generación y manejo de estos datos difieran en gran medida a lo largo del tiempo.

Diversos términos y disciplinas que están despertando gran interés en el ámbito de las grandes organizaciones, tales como: Gestión de las Relaciones con los Clientes (*CRM*)[Kher00], Gestión del Conocimiento (*Knowledge Managment*)[Wu00], Integración de Aplicaciones (*Enterprise Application Integration*)[Gold99], Portales Corporativos (Enterprise Information Portals)[MERR98] o Comercio electrónico Negocio a Negocio (*Business to Business Electronic Commerce*), tienen en común la necesidad de trabajar sobre visiones unificadas de datos que se encuentran dispersos en distintos puntos de la organización.

Los siguientes subapartados describen estas disciplinas en más profundidad y analizan sus necesidades de técnicas de integración de información distribuida.

II.4.2.1. GESTIÓN DE LAS RELACIONES CON CLIENTES

Las modernas organizaciones se enfrentan a nuevos retos de competencia. Los analistas de negocio y mercado han señalado que en los últimos años el nivel de exigencia de los clientes se ha elevado, y han establecido que la mejora constante de la atención al cliente es y será un criterio fundamental en la determinación del éxito o fracaso de los objetivos comerciales de una corporación.

Por ello, en los últimos años se ha despertado un gran interés por las políticas de "Gestión de las Relaciones de los Clientes" (*Customer Relationship Management*)[Kher00]. Estas políticas pueden resumirse en la sencilla idea de que proporcionar a sus clientes el mejor servicio posible depende en gran medida de realizar un seguimiento constante de las relaciones con los mismos. Sin embargo, la realización de este seguimiento exige el procesamiento de toda la información que sea posible recopilar sobre la interacción entre la organización y los clientes. Sin embargo, esta suele encontrarse dispersa en almacenes de datos en diferentes departamentos y secciones de la corporación. Por ello, la integración de estos datos es un requisito necesario para la puesta en marcha de estas políticas.

El enfoque utilizado mayoritariamente hasta la fecha se basaba en la idea de la construcción de algún tipo de almacén de datos (*data warehouse*) donde se cargan periódicamente grandes volúmenes de datos unificados, que son utilizados como base para procesos posteriores.

Sin embargo, este enfoque no permite realizar procesamientos en tiempo real, por lo que es inadecuado para muchos procesos requeridos. Por ejemplo, una pregunta tan habitual cómo: "¿En qué fase se encuentran en este mismo momento los pedidos abiertos de un cliente?", no puede ser contestada mediante el enfoque batch inherente a los almacenes de datos.

La otra importante limitación de las técnicas de almacén de datos, esto es su inadecuación para tratar datos semi-estructurados, es también importante, especialmente dado el papel creciente que medios como el WWW juegan en la interacción cliente-proveedor. El uso directo de fuentes de datos semi-estructuradas, puede permitir un acceso sencillo a fuentes más actualizadas y una puesta en marcha más rápida de los servicios de integración de datos.

Por lo tanto, los sistemas mediadores pueden aportar importantes ventajas en este ámbito.

II.4.2.2. PORTALES CORPORATIVOS

Los portales corporativos, también conocidos como EIPs (*Enterprise Integration Portals*)[MERR98], nacen en los últimos 5 años con la intención de aprovechar la creciente madurez de las tecnologías Internet, para proporcionar a cada empleado de una organización un frontal web desde el que acceder a todos los servicios corporativos que sean relevantes para su actividad diaria.

La idea básica consiste en que cada empleado de la empresa dispone de acceso a un portal web en el que, tras autenticarse debidamente, podrá acceder a una serie de servicios personalizados. Estos servicios incluirán fundamentalmente utilidades corporativas que faciliten su trabajo diario y refuercen la adecuación a los procesos y metodologías de la empresa. Por ejemplo, un empleado podría realizar búsquedas en línea sobre colecciones de documentos de toda la empresa, concertar citas, reservar salas de reuniones, publicar informes y documentos para su conocimiento por aquellos a los que puedan ser relevantes, introducir partes de horas, recibir noticias internas a la empresa que puedan ser de su interés, acceder en cualquier momento a los procedimientos y políticas de la empresa, y un largo etcétera de servicios.

De esta manera, la organización dispone de un mecanismo extremadamente eficiente para publicar y recolectar información sobre su funcionamiento interno, así como para agilizar el trabajo de sus empleados.

Para que estas aplicaciones puedan cumplir sus objetivos plenamente, deben ser capaces de distribuir a cada empleado la información que es relevante para él y en un formato adecuado para sus necesidades. Para que esto sea posible, nuevamente es necesario aplicar previamente técnicas de integración de información.

Debido a que en la mayoría de estos entornos existen grandes volúmenes de información disponible que tienen una estructura débil, habitualmente se han empleado técnicas de integración de datos no estructurados. Sin embargo, esta aproximación desaprovecha la estructura presentada por un amplio subconjunto de esta información, que encaja mejor con la abstracción de datos semi-estructurados, lo que les convierte en aptos para tratamientos más sofisticados basados en la realización de consultas precisas sobre los mismos. Por ejemplo, prácticamente todos los documentos intercambiados entre los distintos empleados de una empresa, guardan algún tipo de

estructura (que normalmente vendrá marcada por los procesos y metodologías internos de la organización), que las técnicas de integración no estructurada desaprovechan totalmente, permitiendo tan sólo una búsqueda por palabra clave, claramente insuficiente para muchos propósitos.

Los sistemas mediadores pueden, por tanto, aportar a los portales corporativos una mayor flexibilidad en su construcción, así como la posibilidad de tratamientos de datos más sofisticados y adecuados a las necesidades productivas de la organización.

II.4.2.3. INTEGRACIÓN DE APLICACIONES EMPRESARIALES

Es evidente que las grandes organizaciones de hoy en día dependen decisivamente de sus aplicaciones informáticas para la realización de sus procesos de negocio.

Factores como la rápida evolución tecnológica, las presiones del mercado, la distribución de los entornos de trabajo o el efecto de la competencia entre diferentes proveedores de soluciones técnicas, ha causado que los entornos, tecnologías y aplicaciones utilizadas provengan de diferentes fabricantes y muestren un alto grado de heterogeneidad.

Esta situación desemboca de manera casi inevitable en escenarios en los que:

1) No es posible coordinar el funcionamiento de las diferentes aplicaciones para que estas se comporten de acuerdo a una estrategia global. Esto, en la práctica, limita severamente la capacidad real de los gestores para introducir cambios o mejoras en sus estrategias, dada la importancia de las tareas desempeñadas por estas aplicaciones, y la dependencia que la organización tiene de ellas.

2) Los resultados de las diferentes aplicaciones no pueden ser fácilmente integrados entre sí. Esto quiere decir que los gestores se ven obligados a trabajar con datos incompletos.

3) Las respuestas a la mayor parte de las preguntas que pueden ser de interés para los gestores no pueden ser contestadas fácilmente, ya que requieren de la integración de procesos y datos de diversas aplicaciones que son incompatibles entre sí. Preguntas que pueden parecer tan simples y básicas como: "¿cuáles son hasta el momento los resultados económicos de la organización en el cuatrimestre actual?", pueden requerir para su contestación del lanzamiento de múltiples procesos manejados por múltiples aplicaciones y cuyos datos, en formatos incompatibles, deben ser integrados. Por ello, muchas grandes organizaciones de la actualidad no pueden contestar a esa pregunta en tiempo real.

Todos estos problemas no pueden ser solucionados solamente mediante la aplicación de técnicas de integración de información, sino que son necesarias técnicas adicionales. Sin embargo, no cabe duda de que las técnicas de Integración de Información son una pieza clave en la tarea.

Este problema es abordado en su totalidad por la disciplina de Integración de Aplicaciones Empresariales (EIA). En [Gold99] se propone un marco de trabajo que permite dividir esta compleja tarea en capas de referencia, de manera que sea más fácil encajar software de diferentes fabricantes y con diferentes funcionalidades. Una de las capas de esta arquitectura se refiere específicamente a la Integración de Datos distribuidos.

II.4.2.4. COMERCIO ELECTRÓNICO NEGOCIO A NEGOCIO

El comercio electrónico, entendido como el uso de las tecnologías telemáticas para automatizar todos o parte de los procesos involucrados en las operaciones de negocio, lleva siendo utilizado desde hace décadas en sectores como el financiero, como una manera de abaratar sus costes de procesamiento.

Sin embargo, hasta la aparición de Internet unos pocos años atrás, poner en marcha este tipo de soluciones involucraba el establecimiento de redes privadas entre los agentes comerciales participantes. Esto hacía su coste inaccesible para la gran mayoría de organizaciones.

La aparición de Internet ha proporcionado a las organizaciones un mecanismo de interconexión global y de bajo coste, que ha disparado su interés por poner en marcha soluciones de comercio electrónico que la liguen de una manera más eficiente con sus proveedores, clientes y socios, beneficiándose así de la reducción de costes y sobrecarga operativa asociada.

Sin embargo, el éxito de estas operaciones está llevando a mucha organizaciones a perseguir un grado de integración todavía mayor, con el objetivo de aumentar su competitividad. Esto obliga a las aplicaciones de cada empresa participante a integrarse cada vez más estrechamente entre sí.

Sin embargo, estas aplicaciones son heterogéneas, y presentan un alto grado de autonomía, por lo que su integración puede ser extremadamente compleja. Además, por razones tales como el secreto industrial, cada organización exigirá mantener un control muy estricto sobre sus datos y aplicaciones, por lo que una solución homogeneizadora no es posible en la práctica.

Nuevamente, pues, nos encontramos con la necesidad de integrar fuentes de datos dispersas, heterogéneas y con un grado muy alto de autonomía. Además, el frontal ofrecido por una organización de cara al exterior es hoy, fundamentalmente, un frontal web de características semi-estructuradas.

Por lo tanto, los sistemas mediadores pueden colaborar, junto con las técnicas de Integración de Aplicaciones Empresariales, para proporcionar una manera más rápida, flexible y mucho menos intrusiva, de integrar los diferentes frontales y aplicaciones de los participantes, evitando costosos desarrollos ad-hoc de débil escalabilidad y

permitiendo que cada organización mantenga el control autónomo de sus datos y sistemas.

II.4.2.5. GESTIÓN DEL CONOCIMIENTO

En los últimos años ha tomado auge una línea de pensamiento por la cual, el principal capital de una empresa y su principal valor, es el conocimiento explícito o implícito que contiene.

Este conocimiento puede tomar muchas formas, no todas ellas evidentes; puede ser explícito o implícito, estar almacenado en algún repositorio o residir solamente en personas concretas, etc.

Las estrategias de Gestión del Conocimiento se basan en primer término en identificar y estructurar todo el conocimiento relevante de la organización. Posteriormente, ese conocimiento debe ser difundido en la organización, tratando así de que pueda beneficiar a todas sus áreas y no quedar confinado en su lugar de origen. También se pretende que facilite un mayor grado de coordinación entre los diferentes componentes de la misma.

No es difícil darse cuenta que las técnicas de integración de información pueden ser una pieza clave en la puesta en marcha de una estrategia de este tipo. Por lo tanto, nuevamente los sistemas mediadores pueden aportar su flexibilidad para integrar con poco esfuerzo fuentes estructuradas y semi-estructuradas, y construir con poco esfuerzo aplicaciones que los utilicen para los propósitos específicos que la organización desee.

□

Por lo tanto, como se ha visto, todas estas estrategias y aplicaciones empresariales precisan de una manera u otra de técnicas de Integración de Información.

El enfoque basado en Almacenes de Datos, si bien es adecuado para la realización de procesos batch tales como aquellos que requieren algún tipo de Minería de Datos sobre históricos, muestra un importante número de limitaciones tales como: 1) Debido a que los datos no se encuentran actualizados, no es adecuado para aplicaciones que requieran funcionamiento en tiempo real (tales como las que son frecuentes en, por ejemplo, la disciplina de Integración de Aplicaciones) 2) Los sistemas construidos hasta el momento no tienen capacidad para aprovechar las regularidades presentes en los datos semi-estructurados, y tienen que tratarlos como si estuviesen carentes de estructura 3) Debido a que requiere de la construcción de un gran repositorio de datos y de los procesos de migración de datos desde las fuentes originales, su puesta en marcha es larga y costosa 4) No es valido en la práctica para soportar repositorios inter-organización como aquellos necesarios en aplicaciones de Comercio Electrónico de Negocio a Negocio.

Los sistemas mediadores pueden aportar un nivel de flexibilidad mucho mayor: pueden ponerse en marcha de manera mucho más rápida, no son necesarios grandes repositorios centrales, ni diseñar complicados procesos de migración y actualización de datos, permiten tratar los datos semi-estructurados de una manera similar a si fuesen estructurados, permiten la operación en tiempo real, pueden manejar fuentes con un muy alto grado de autonomía, etc.

Por ello, no es extraño que actualmente haya un gran interés por la construcción de sistemas mediadores adecuados a este tipo de entornos.

II.5. ARQUITECTURA DE UN SISTEMA MEDIADOR

En este apartado se repasará en detalle el funcionamiento interno de un sistema mediador. Se comienza con una descripción general de su arquitectura para posteriormente analizar en más detalle los diferentes enfoques posibles a adoptar a la hora de construir un sistema de este tipo, así como las principales técnicas utilizadas.

En la Figura II.5.1 se muestra la arquitectura típica de un sistema mediador. Las siguientes líneas describen cómo interactúan los componentes de la misma para implementar el flujo típico seguido por una consulta recibida por el sistema (puede consultarse el Ejemplo II.3.4.1 como recordatorio de las principales tareas que esto involucra).

El mediador recibe consultas sobre el esquema mediado (recuérdese que este concepto fue introducido en el apartado II.3.4) expresadas en algún lenguaje de consulta, como por ejemplo SQL.

Una vez recibida la consulta, el Generador de Planes genera los posibles planes de ejecución de la misma. Cada plan estará compuesto por un conjunto de subconsultas a emitir a una serie de fuentes y una serie de labores de post-procesado y unificación que es necesario realizar con los resultados obtenidos de las fuentes para obtener el resultado final. Obviamente, la subconsulta emitida a cada fuente debe ser permitida por las capacidades de consulta de la misma para que el plan de ejecución sea factible.

Una vez generados los posibles planes, el optimizador entra en funcionamiento para escoger el plan de ejecución más óptimo. El plan escogido es enviado al motor de ejecución, que es el encargado de llevarlo a la práctica.

Para la ejecución de las subconsultas sobre las fuentes, el motor de ejecución delega en los programas envoltorio (o *wrappers*). El programa envoltorio de cada fuente tiene como misión proporcionar una visión de la misma similar a la de una tabla en una base de datos convencional. Para ello, debe ser capaz de usar el lenguaje de consulta nativo de la fuente y de obtener los resultados a las consultas efectuadas sobre la misma en un formato que pueda ser procesado por el mediador.

Todo este proceso es auxiliado por la información contenida en el diccionario de datos del mediador, en el que además de la descripción de las relaciones virtuales que componen el esquema mediado, se encuentran también descripciones de las fuentes que el mediador precisa para su funcionamiento, y cuya composición será detallada en el apartado II.5.1.

Si comparamos la arquitectura de un mediador con la de una base de datos tradicional, existen principalmente dos diferencias fundamentales:

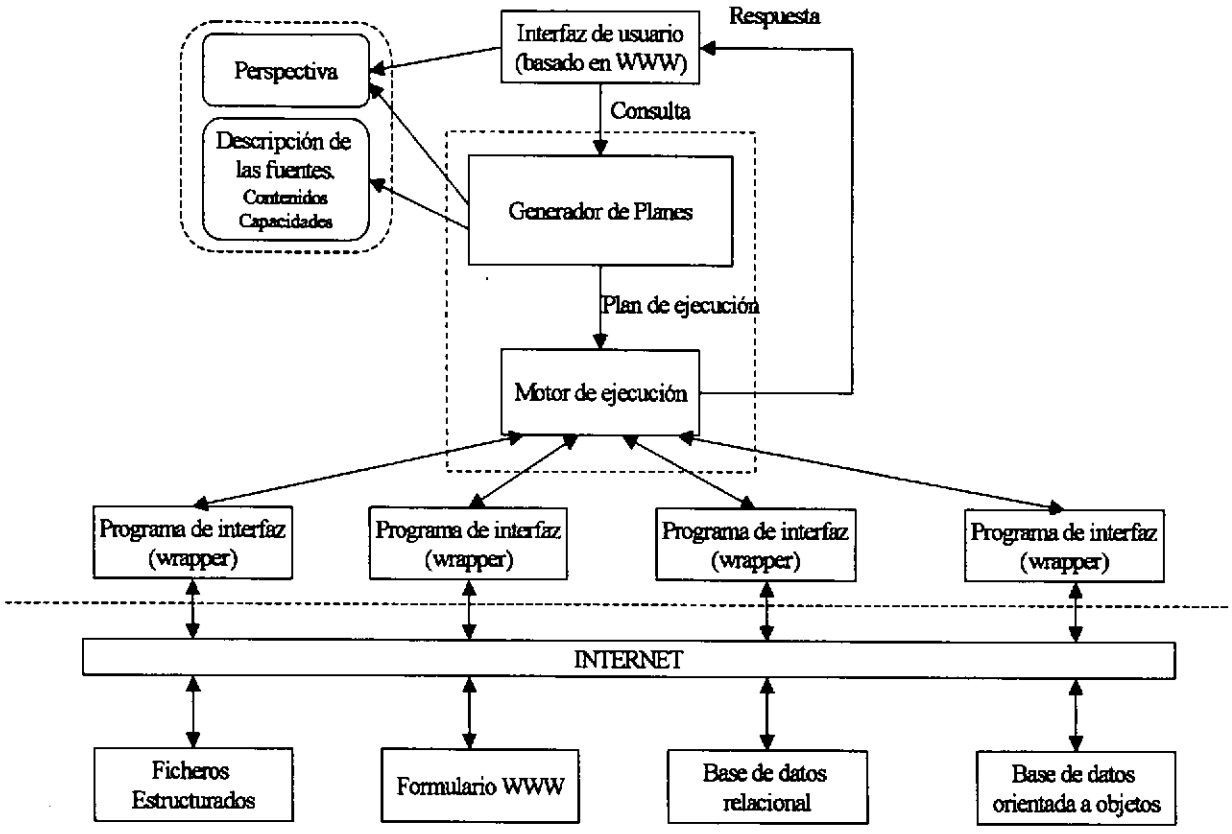


Figura II.5.1: Arquitectura de un sistema de integración de datos virtual (mediador)

- El sistema no accede directamente a los datos sino que lo hace a través de los programas envoltorio, que pasan a ser un punto crucial de la arquitectura de los mediadores.
- El usuario no realiza consultas directamente sobre el esquema en el que están almacenados los datos. La razón de esto es que uno de los principales objetivos de un sistema de integración de datos, es que el usuario no tenga que conocer nada acerca de cada fuente de datos específica. En lugar de ello el usuario realiza las consultas sobre el esquema mediado mantenido por el sistema mediador.

Como ya se ha comentado previamente, existen importantes problemas que deben ser abordados en la construcción de sistemas mediadores. Los principales son:

- El modo de especificación del esquema mediado y la reformulación de las consultas sobre el mismo en subconsultas sobre las fuentes. Esto involucra, entre otras tareas, representar y tratar adecuadamente la información sobre las capacidades de consulta de las fuentes.
- El procedimiento para la ejecución de consultas. Involucra la ejecución eficiente de los planes de consulta y el uso de técnicas de optimización adaptadas al modo específico de funcionamiento de los mediadores.

- La construcción de programas envoltorio. Cómo ya se ha visto, los programas envoltorio son una parte crucial de la arquitectura de los sistemas mediadores. Por lo tanto las técnicas destinadas a su generación y mantenimiento con el menor esfuerzo posible tienen una gran importancia dentro de los sistemas de mediación.
- La heterogeneidad de representación entre las distintas fuentes. En un sistema mediador, las diversas fuentes involucradas pueden utilizar diferentes taxonomías y convenciones de nombrado y representación. Esto puede causar serios problemas a los sistemas mediadores que, por ejemplo, pueden no reconocer la identidad de un mismo objeto en dos fuentes diferentes.

Los siguientes apartados se ocupan en detalle de cada uno de estos problemas, esbozando los principales enfoques que hasta ahora se han adoptado para abordarlos.

II.5.1. REFORMULACIÓN DE LOS ESQUEMAS DE LAS FUENTES EN UN ESQUEMA GENERAL

Como ya se ha comentado en apartados previos, el esquema mediado o esquema global especifica un conjunto de relaciones compuestas de un conjunto de atributos (frecuentemente de un tipo de datos concreto), que son las que utiliza el usuario para escribir sus consultas.

Por lo tanto, para evaluar una consulta sobre el esquema mediado, el mediador debe convertirla en un conjunto de subconsultas sobre los esquemas locales de las fuentes. Para poder realizar este paso de reformulación de consultas y la posterior combinación y unificación de resultados, el mediador necesita descripciones de cada fuente que especifiquen aspectos tales como los siguientes:

- Contenido de la fuente. Es decir, qué representan sus tuplas (por ejemplo referencias bibliográficas, productos, etc.)
- El esquema de la fuente. Por ejemplo en el caso de los libros una fuente podría contener para cada tupla los atributos título, autores y precio, todos ellos de tipo cadena de caracteres.
- Formatos de representación utilizados por la fuente. Por ejemplo, si estamos hablando de una fuente de referencias bibliográficas, qué estructura y formato tiene el atributo autores (es decir, cómo se separan los diferentes autores de un libro, qué formato se usa para representar un nombre completo, etc.)
- Capacidades de procesamiento de consultas. Esto describe qué tipo de consultas pueden realizarse sobre los datos de la fuente. Por ejemplo, una tienda de libros en línea puede permitir acceder a su catálogo sólo mediante la realización de selecciones de sus tuplas por título o autor.
- Información de costes que permita construir planes de ejecución de consultas tan óptimos como sea posible. Esa información puede utilizarse para obtener mejores planes de ejecución.

En función de estas descripciones, dada una consulta sobre el esquema mediado, el mediador debe elaborar un plan de ejecución que cumpla las siguientes características:

- Ser *correcto*. Esto es que devuelva los resultados correctos para la consulta.
- Ser *factible*. Es decir que esté compuesto de subconsultas a las fuentes que éstas estén capacitadas para responder.
- Ser *eficiente*. Es decir que intente minimizar el coste de la consulta.

En la actualidad existen numerosos trabajos de investigación centrados en el problema de cómo especificar las descripciones de las fuentes y cómo usar después esas descripciones para realizar las reformulaciones de las consultas. Se han propuesto dos formas generales para abordar el problema: *Global as view* (GAV) o definición de vistas y *Local as view* (LAV) o definición de fuentes.

La aproximación de definición de vistas (GAV), que normalmente cuenta con el respaldo de la comunidad de Bases de Datos, consiste básicamente en describir las relaciones del esquema mediado como vistas sobre las relaciones exportadas por las fuentes base. La aproximación de definición de fuentes (LAV), que se originó en las comunidades de inteligencia artificial y lógica descriptiva, asume la existencia de predicados y colecciones (o relaciones) globales, y define los contenidos de las fuentes a partir de dichos predicados. Los subapartados II.5.1.1 y II.5.1.2 se ocupan de cada una de ellas en detalle.

El tema de cómo tratar las limitaciones en las capacidades de consulta se estudia en mayor detalle en el subapartado II.5.1.3

II.5.1.1. ENFOQUE DE DEFINICIÓN DE VISTAS (GLOBAL AS VIEW)

En el enfoque GAV (seguido por sistemas como [GMPQ95][HKWY97][ACPS96][[TRV98]]), para cada relación R en el esquema mediado, se define una vista sobre las relaciones de las fuentes, especificando cómo obtener las tuplas de R. Una vista se define de la misma manera y con la misma sintaxis con la que se escriben las consultas (por ejemplo, álgebra relacional).

De esta manera, finalmente, cada relación del esquema mediado podrá expresarse mediante un árbol de operaciones algebraicas cuyas hojas son las relaciones de las fuentes base.

Así, una vez definidas las relaciones del esquema mediado y dada una consulta sobre dicho esquema, el mediador puede "expandir" la consulta, sustituyendo las referencias a las fuentes del esquema global por su expresión algebraica en función de las relaciones de las fuentes base. De esta manera, se producirá una reescritura de la consulta que haga referencia únicamente a las relaciones base.

Ejemplo II.5.1.1: Reescritura en enfoque GAV

Supóngase un sistema mediador con tres fuentes `A(X,Y,Z)`, `B(X,Y,Z)` y `C(Y,K)`. Supóngase que una de las relaciones del esquema mediado viene definida como `R(X,Y,Z,K)` = $(A \cup B) \infty_y C$.

Supóngase ahora que el mediador recibe la consulta expresada en álgebra relacional:

$\sigma_{(x=x0 \text{ and } k=k0)}(R)$

que en lenguaje sql se escribiría:

```
select * from r where x=x0 and k=k0
```

El mediador empezaría el proceso de despliegue de vistas sustituyendo R por su expresión y obteniendo así:

(i) $\sigma_{(x=x0 \text{ and } k=k0)}(R) = \sigma_{(x=x0 \text{ and } k=k0)}((A \cup B) \infty_y C))$
(ii) $= (\sigma_{x=x0}(A) \cup \sigma_{x=x0}(B)) \infty_y \sigma_{k=k0}(C)$

Un posible plan de consulta, por lo tanto, será el siguiente:

(i) Se envía a `A` la subconsulta $\sigma_{x=x0}(A)$, a `B` la subconsulta $\sigma_{x=x0}(B)$ y a `C` la subconsulta $\sigma_{k=k0}(C)$. Los envoltorios de `A`, `B` y `C` son los responsables de efectuar dichas subconsultas en las fuentes y devolver los resultados al mediador.

(ii) El mediador realiza la unión de los resultados devueltos por `A` y `B` y hace el join del resultado de la unión con los resultados devueltos por `C`. El resultado obtenido es la respuesta a la consulta.

Así que, como puede verse, la reformulación de consultas en el enfoque GAV es simple, ya que se reduce a ir "desplegando" las vistas que constituyen las relaciones del esquema mediado de acuerdo al árbol que describe como se construyen.

La principal ventaja de esta alternativa es que las fuentes base pueden combinarse de maneras tan complejas como se desee sin que por ello se complique ni la definición del esquema mediado ni la ejecución de las consultas sobre el mismo.

Una importante ventaja adicional del enfoque GAV es que, debido a que el procesamiento de consultas es similar al de los sistemas de Bases de Datos relacionales, la creación de motores de ejecución para los mediadores puede aprovecharse, al menos en parte, de la experiencia acumulada con la realización de los motores de ejecución ya existentes. Otra ventaja relacionada es que, debido a su similitud con el proceso de definición de vistas en una base de datos convencional, cualquier administrador de bases de datos puede formarse rápidamente como

administrador de un sistema mediador, lo que incrementa la facilidad de integración en la industria de este tipo de sistemas.

Por contra, este enfoque presenta el inconveniente de su falta de modularidad, ya que cada vez que se añade una nueva fuente al sistema, habrá que cambiar la definición de las relaciones del esquema mediado si queremos que contemplen a la nueva fuente. Esto viene derivado de que no existen descripciones directas de las contribuciones de cada fuente a las relaciones globales.

Ejemplo II.5.1.2: Falta de modularidad en el enfoque GAV

Supongamos que en el ejemplo anterior las fuentes A y B representaban los catálogos electrónicos accesibles en línea de los proveedores de una determinada organización y que C era una base de datos con valoraciones de los productos ofertados. R sería en este caso una relación unificada con los productos de A y B junto con sus valoraciones según C.

Supongamos ahora que se desea añadir un nuevo proveedor al sistema D. Es necesario cambiar la definición de R para contemplar a la nueva fuente, siendo la nueva expresión $R(X,Y,Z,K) = (A \cup B \cup D) \infty_{y} C$

□

De todas maneras, salvo que el ritmo al que se añaden las fuentes al sistema sea muy alto, la necesidad de redefinir las relaciones del esquema mediado no parece un grave inconveniente para la mayoría de aplicaciones, sobre todo si las maneras en que estas se combinan no cambian al añadir la fuente (como, por ejemplo, en el caso anterior).

II.5.1.2. ENFOQUE DE DEFINICIÓN DE FUENTES (LOCAL AS VIEW)

La alternativa LAV (utilizada por sistemas como[DG97][LRO96] [KMAA98]) está enfocada desde el punto de vista opuesto a la GAV: para cada fuente de información S, se escribe una consulta sobre las relaciones en el esquema mediado que describe qué tuplas se pueden encontrar en S. Es decir, asume la existencia de predicados y colecciones globales (que constituirán las relaciones del esquema mediado), y define los contenidos de las fuentes a partir de ellos.

Ejemplo II.5.1.3: Enfoque LAV

Supóngase un esquema mediado construido para almacenar información sobre películas, con una relación P(TIT, DIR, ACT, NAC), cuyos atributos representan, respectivamente, el título, director, actor principal y nacionalidad de la película. Supóngase también una fuente A que sólo contuviese películas españolas.

En un enfoque LAV podríamos definirla especificando que de ellas se obtienen tuplas de `P` que cumplen el predicado `nac=española`, siendo `española` uno de los posibles valores definidos en `P` para el atributo `nac`. Una notación habitual para expresar esto sería:

```
v (TIT,DIR,ACT,NAC) :- P (TIT,DIR,ACT,española)
```

donde `v` sería una regla de definición de la fuente `A`.
□

Es importante notar que esta definición no puede ser considerada exactamente como una definición de vista "a la inversa" (por vista "a la inversa" se entendería aquí definir las relaciones de las fuentes como una vista sobre las relaciones del esquema mediado, justo a la inversa que en el enfoque GAV). Nótese que la definición no implica que todas las películas con nacionalidad española se encuentren en esa fuente y que, por lo tanto, la consideración de esta definición como una vista sería errónea.

En la aproximación LAV es más fácil añadir o eliminar fuentes debido a que en sus descripciones no se tiene en cuenta la posible interacción con otras fuentes y también es más fácil describir restricciones en los contenidos de las mismas.

Ejemplo II.5.1.4: Modularidad del enfoque LAV

Supóngase que en el ejemplo anterior se añade una nueva fuente con información sobre películas norteamericanas. La nueva fuente puede describirse sencillamente con la regla:

```
v (TIT,DIR,ACT,NAC) :- P (TIT,DIR,ACT,estadounidense)
```
□

En este caso, el problema de la reformulación es análogo al problema de responder consultas usando vistas. Una buena recopilación de las principales ideas y avances en este activo campo de investigación puede encontrarse en [Lev00].

Este problema es significativamente más complejo que en el caso GAV. Además los algoritmos de reformulación son no deterministas incluso con lenguajes de consulta relativamente sencillos. Esto se traduce en que en el enfoque LAV, la ejecución de consultas y propagación de capacidades son más complejos y menos eficientes. Además los lenguajes de descripción de fuentes y de realización de consultas tienen que ser más simples, ya que de lo contrario el problema de la eficiencia y el no determinismo de los algoritmos de reformulación podría agravarse sobremanera. Para entender bien esto, es conveniente un estudio en más detalle de las interioridades de los métodos LAV. El siguiente subapartado proporciona dicha visión.

II.5.1.2.1. EJEMPLO DE REFORMULACIÓN EN SISTEMAS LAV

Para ilustrar la idea básica seguida por estos sistemas en su proceso de reformulación de consultas, y que puedan entenderse con mayor claridad sus debilidades y fortalezas, considérese el siguiente ejemplo tomado (con ligeras modificaciones) de [Ull97].

En primer lugar, se definen algunos conceptos que serán necesarios para la correcta comprensión del ejemplo.

Tal y como es habitual en sistemas LAV, a lo largo de todo el ejemplo se utilizará una notación [ULL88] basada en reglas lógicas del tipo:

```
p (X,Z) :- a (X,Y) & a (Y,Z)
```

donde `a` es un predicado *extensional* (esto es, un predicado *base*, no obtenido a partir de reglas lógicas) y `p` es un predicado *intensional* (esto es, obtenido a partir de otras reglas lógicas).

La parte izquierda de la regla se llama la *cabeza* de la misma y contiene un sólo átomo y el símbolo condicional '`:-`'. En la parte derecha (*cuerpo* de la regla) hay cero o más átomos, que se llaman las *submetas* de la regla. En la cabeza siempre hay un predicado *intensional* (por definición). En el cuerpo puede haber predicados intensionales o extensionales.

Las reglas expresan una relación condicional entre el cuerpo y la cabeza de la regla. Por ejemplo, la regla anterior se leería: " si `a(X,Y)` es cierto y `a(Y,Z)` es cierto, entonces `p(X,Z)` también lo es".

Un tipo especial de reglas que será interesante para nuestros propósitos son las llamadas *consultas conjuntivas*. Una *consulta conjuntiva* es una regla de inferencia lógica cuyas submetas están todas formadas por predicados extensionales.

Una consulta conjuntiva se *aplica* a las relaciones o predicados extensionales, cogiendo todos los valores ciertos de los mismos y asignándolos a las variables del cuerpo de la regla. Aquellas sustituciones que hagan ciertas todas las submetas, se aplican a la cabeza de la regla y constituyen un nuevo hecho inferido para el predicado de la cabeza de la regla.

En nuestro caso, al igual que en la mayoría de los sistemas LAV construidos hasta la fecha, las consultas conjuntivas constituirán, como se verá más adelante, tanto el lenguaje en el que se expresarán las consultas sobre el esquema mediado como el lenguaje que se utilizará para realizar la descripción del contenido de las fuentes.

Una cuestión importante acerca de las consultas conjuntivas es saber cuando una de ellas está *contenida* en otra. Si `q1` y `q2` son consultas conjuntivas, entonces diremos

que q1 contiene a q2 (y lo denotaremos como $q1 \subseteq q2$) si para todas las bases de datos D (entendiendo por base de datos, una asignación de valores a las variables que sea cierta para los predicados extensionales), el resultado de aplicar q1 a D (escrito q1(D)) es un subconjunto de q2(D).

Además diremos que dos consultas conjuntivas son *equivalentes* si y sólo si ambas se contienen mutuamente.

Supóngase ahora un sistema que contiene información sobre los empleados de una empresa. Dicha información está contenida en tres fuentes. La primera de ellas, S1, contiene información sobre el teléfono y el jefe directo de los empleados. La segunda de ellas, S2, contiene información acerca de la oficina y el departamento de los empleados. Finalmente la tercera contiene información sobre teléfonos, pero sólo de empleados pertenecientes al departamento de juguetes. En el enfoque LAV, esta situación puede representarse de la siguiente manera. Durante todo el ejemplo se utiliza notación Datalog[Ull88].

Se definen en primer lugar los predicados globales (equivalente a las relaciones del esquema mediado). En este caso, una posible lista sería:

Emp(E), significa que E es un empleado.
Teléfono(E,T), significa que T es el teléfono de E.
Oficina(E,O), significa que O es la oficina de E.
Jefe(E,J), significa que J es el jefe directo de E.
Departamento(E,D) significa que D es el departamento de E.

Una vez definidos los predicados globales, los contenidos de una fuente se describen mediante una o más consultas conjuntivas, considerando a los predicados globales como extensionales.

Así, la descripción de las tres fuentes en términos de los predicados globales sería:

```
s1 (E,T,J) :- emp(E) & telefono(E,T) & jefe(E,J)
s2 (E,O,D) :- emp(E) & oficina E,O) & departamento(E,D)
s3 (E,P) :- emp(E) & telefono(E,T) & departamento(E,juguetes)
```

Es importante resaltar que no se asume que las fuentes sean completas. Esto quiere decir que es posible, por ejemplo, que las fuentes no contengan el teléfono de todos los empleados. Tampoco se asume nada sobre la consistencia entre fuentes: por ejemplo, vistas las descripciones de las fuentes, los empleados del departamento de Juguetes podrían aparecer tanto en la fuente 1 como en la 3. Si fuese así, es posible que ambas informaciones fuesen inconsistentes. El sistema no supondrá nada al respecto.

Una vez definidos los predicados globales y las descripciones de las fuentes, el sistema puede recibir consultas sobre los predicados globales. Por ejemplo, el sistema podría recibir una consulta preguntando el teléfono y la oficina de un determinado

empleado, al que llamaremos 'Juan'. Esta consulta *q* se expresaría de la siguiente forma:

```
q(T,O) :- telefono(Juan,T) & oficina(Juan,O)
```

Para reescribir esta consulta en función de las fuentes y obtener la respuesta, el sistema debe buscar lo que se llama una *solución mínima* para q. Esa será la salida del proceso de reformulación.

Para entender el concepto de solución mínima, veamos primero el de solución. Una *solución* S para q es una expresión lógica (también del tipo *consulta conjuntiva*) escrita en función de las descripciones de las fuentes (s1, s2 y s3 en este caso) tal que, cuando se sustituyen las fuentes que aparecen en la expresión de S por sus definiciones en función de los predicados globales, se obtiene una expresión (llamada *consulta expandida* y que denotaremos por E) equivalente a q.

Si no existiese una *solución* S equivalente a q, se considerará también como válida la solución obtenida de realizar la unión de todas aquellas soluciones parciales cuya consulta expandida E, esté contenida en q.

Una *solución* S para Q es *mínima* si:

1. $S \subseteq Q$
2. No existe una solución T para Q tal que:
3. $S \subseteq T \subseteq Q$, y
4. T tiene menos submetas que S.

Intuitivamente, una *solución mínima* para q es una consulta conjuntiva expresada en función de las fuentes que o bien es equivalente a q, o bien es la reescritura en función de las fuentes más cercana posible a la equivalencia con q.

La razón por la que no siempre habrá una reescritura que sea equivalente a la consulta original es que las fuentes pueden no disponer de contenidos que lo permitan o que, aún disponiendo de los mismos, el lenguaje utilizado en su descripción (consultas conjuntivas en este caso) no sea lo suficientemente expresivo para permitir definirlos bien (aunque se hará hincapié en el tema más adelante, nótese que las consultas conjuntivas tienen fuertes limitaciones como lenguaje de consulta).

Existen varios métodos para hallar soluciones mínimas. Todos ellos consisten en generar posibles reescrituras de la consulta recibida e ir probando para cada una de ellas si está contenida en Q. Para cada reescritura S que lo esté, se comprueba además si Q está contenida en S. Si esto fuese así, se habría encontrado una reescritura de la consulta equivalente a Q y el proceso puede detenerse. Si no es así, se siguen buscando más soluciones para Q y finalmente, de entre todas ellas, se devuelve un conjunto de soluciones mínimas cuya unión está contenida en Q. Si se termina la búsqueda de

posibles reescrituras sin que ninguna esté contenida en Q, entonces no se obtiene solución para la consulta.

Por ejemplo, para nuestro caso, existen dos soluciones mínimas para q. Ambas usan a s2 para conseguir la oficina de Juan. Difieren en que, para conseguir su teléfono, una de ellas usa s1 y la otra s3. Vienen expresadas por las siguientes consultas conjuntivas:

```
respuesta (T,O) :- s1(Juan,T,M) & s2(Juan,O,D)
respuesta (T,O) :- s3(Juan,T) & (s2(Juan,O,D)
```

Por lo tanto, el sistema mediador ejecutará ambas reescrituras sobre las fuentes y devolverá como respuesta la unión de los resultados de ambas.

Vale la pena volver a recalcar que las consultas expandidas de estas reescrituras (que, como ya se ha dicho, se obtienen sustituyendo en la expresión las fuentes por su descripción en función de los predicados globales):

```
respuesta(T,O):-emp(Juan)&telefono(Juan,T)&jefe(Juan,J)&emp(Juan)&
    oficina(Juan,O)&departamento(Juan,D)
respuesta(T,O):-emp(Juan)&telefono(Juan,T)&departamento(Juan,
Juguetes)&emp(Juan)&oficina(Juan, O)&departamento(Juan, D)
```

no son equivalentes a q: son sólo las consultas conjuntivas contenidas en q que están más cercanas a q. Intuitivamente, es la respuesta más aproximada a q que puede darse con esas fuentes y esas descripciones de sus contenidos. Nótese que podría ocurrir que con los mismos contenidos en las fuentes, pero con un lenguaje de descripción de fuentes más expresivo que las consultas conjuntivas, se pudiesen realizar descripciones más ricas de los contenidos de las fuentes, haciendo posible encontrar soluciones más cercanas a q.

El problema principal a la hora de crear un algoritmo que realice los pasos anteriores es cómo acotar el espacio de búsqueda de posibles reescrituras para que el proceso sea eficiente. Se ha demostrado que este problema, utilizando las consultas conjuntivas como lenguaje de consulta y descripción de fuentes, es NP-completo[Lev95]. Se han propuesto diversos métodos para realizar esa acotación. Los principales son el algoritmo del cubo[LRO96], el algoritmo de las reglas inversas[DG97b], el algoritmo Mini-Con[PL00] y el algoritmo CoreCover[ALU01]. Han empezado también a aparecer algoritmos que abordan el problema de reescritura con lenguajes de consulta más ricos que las consultas conjuntivas, aunque obteniendo métodos de complejidad algorítmica mayor. Por ejemplo, [CAL00]estudia el problema utilizando lenguajes de consulta para datos semiestructurados.

Como conclusiones finales, podemos extraer que los métodos LAV son significativamente más complejos que los GAV y sus algoritmos de reformulación serán normalmente menos eficientes. Además, las maneras en que se pueden combinar las fuentes están limitadas por la capacidad expresiva de lenguajes relativamente poco

potentes, como el de las consultas conjuntivas. Esto es así, porque utilizando lenguajes más expresivos, la complejidad algorítmica de la obtención de reescrituras podría hacer que el sistema fuese muy ineficiente.

Otra ventaja de GAV es su similitud con la definición de vistas en sistemas de bases de datos convencionales, lo que facilita su inserción en la industria.

A cambio, los sistemas LAV ofrecen un nivel de modularidad superior a los sistemas GAV (lo que facilita la adición de fuentes), y permiten expresar de manera muy natural las restricciones en los contenidos de las fuentes (por ejemplo, nótese la manera simple y elegante en que se pudo representar en el ejemplo que `s3` sólo dispone de teléfonos de empleados del Departamento de Juguetes, algo que un sistema GAV tendría que tratar con algún mecanismo específico). Además la expresividad de sus lenguajes de consulta y su nivel de eficiencia es más que suficiente para muchas aplicaciones.

Por lo tanto, el enfoque más adecuado a seguir depende de las características de las aplicaciones que vayan a funcionar sobre el sistema mediador, sin que se pueda decir claramente que un enfoque es superior al otro.

II.5.1.3. DIFERENCIAS EN LAS CAPACIDADES DE PROCESAMIENTO DE CONSULTAS

Una Base de Datos convencional expone una serie de relaciones sobre las que es posible realizar consultas por todos sus atributos y utilizando cualquier predicado de selección. No hay, por tanto, ninguna restricción sobre las consultas a realizar, más allá de las que marque la capacidad expresiva del lenguaje de consulta utilizado.

Esto es posible porque el sistema de base de datos tiene un acceso directo a todos los datos contenidos en las relaciones y puede manipularlos directamente sin ninguna restricción.

Esto cambia en el caso de los sistemas mediadores. Como ya se ha comentado previamente, en estos sistemas el acceso a los datos se produce a través de las interfaces proporcionadas por las fuentes para ese efecto. Estas interfaces pueden incluir restricciones sobre las consultas que pueden ser respondidas.

Los principales motivos por los que existen estas limitaciones son:

- Los datos de la fuente pueden estar almacenados en ficheros estructurados o en un sistema heredado, y en este caso la interfaz hacia estos datos está limitada inherentemente por la naturaleza de su modo de almacenamiento.
- Aunque los datos estén almacenados en una Base de Datos relacional, la fuente puede proporcionar solamente una serie de capacidades de acceso limitadas por razones de seguridad o rendimiento. Por ejemplo, en una fuente web que exponga

una interfaz para consultar un catálogo en línea de libros, frecuentemente se nos permitirá realizar la consulta solamente por los atributos 'título' o 'autor' de la relación. Esto quiere decir, por ejemplo, que, al menos directamente, no se pueden hacer consultas que involucren al atributo 'precio'. Siguiendo con el ejemplo, frecuentemente tampoco estará permitido obtener una lista completa de todos los libros del catálogo (es decir, el equivalente a una consulta del estilo '`select * from relacion`') ya que es obligatorio especificar en el formulario de consulta al menos el título o el autor de las referencias buscadas.

Para construir un sistema mediador, es necesario que estas capacidades y limitaciones estén explícitamente descritas en el sistema, y que éste las explote al máximo para generar planes de consulta factibles y lo más óptimos que sea posible. Para realizar estas descripciones es necesario proporcionar dos tipos de información que suelen denominarse *capacidades negativas* y *capacidades positivas*.

Las capacidades *negativas* se usan para expresar restricciones en las consultas que pueden efectuarse sobre una fuente.

La forma que más habitualmente toman estas restricciones es la obligación de que la consulta recibida incluya algún predicado de selección (i.e. las condiciones incluidas en la cláusula WHERE de la subconsulta) involucrando algún atributo determinado.

Ejemplo II.5.1.5: Capacidades de consulta negativas

Normalmente los formularios de consulta en fuentes web (e.g. una tienda electrónica de libros exportando una relación `R={TITULO,AUTOR,PRECIO}`) suelen obligar a que al menos uno de sus campos (e.g. el título del libro a buscar) sea rellenado para que la búsqueda pueda realizarse, lo cuál se traduce en que una subconsulta que vaya a ser ejecutada a través de ese formulario debe por fuerza incluir al menos un predicado de selección que incluya el atributo en cuestión (e.g. `select * from R where TITULO = 'Ulises'`) para que el valor exigido para el atributo por el predicado de selección (e.g. `'Ulises'`) pueda ser utilizado para rellenar el campo y ejecutar la consulta. Una consulta que no incluyese un predicado como ese (e.g. `select * from R`) no podría ser ejecutada por la fuente ya que no hay ningún valor que pueda utilizarse para rellenar el campo obligatorio (e.g. `TITULO`). Nótese que cualquier consulta que incluyese otros predicados de consulta adicionales (e.g. `select * from R where TITULO = 'Ulises'and PRECIO < 5000`) satisfaría también las restricciones impuestas por la fuente, incluso aunque el formulario de consulta no incluyese ningún campo para los atributos adicionales (e.g. `PRECIO`) ya que el mediador siempre podría ejecutar una subconsulta que involucre sólo al atributo obligatorio que aparece en el formulario (e.g. `select * from R where TITULO = 'Ulises'`) y posteriormente filtrar los resultados obtenidos de la fuente aplicando él mismo los predicados de selección adicionales (e.g. `PRECIO < 5000`).

□

Por lo tanto, las *capacidades negativas* representan las restricciones que deben cumplir las consultas recibidas por una fuente para que ésta sea capaz de ejecutarlas. Un plan de consultas generado por el mediador sólo será realizable si todas las subconsultas incluidas en el plan cumplen las restricciones especificadas por las capacidades negativas de la fuente sobre la que deben ejecutarse. Así, ayudan al mediador a generar planes de consulta *factibles*.

Las *capacidades positivas*, por el contrario, permiten representar qué consultas puede ejecutar la fuente *por sí misma* una vez que las restricciones descritas por las capacidades negativas han sido satisfechas. Es importante notar que no se incluyen entre las capacidades de consulta positivas de una fuente aquellas que requieren de post-filtrados por parte del mediador. Así, en el ejemplo anterior, la capacidad de consultar la fuente por el atributo `PRECIO` no se encontraría entre las capacidades de consulta positivas de la fuente. Sin embargo, si el formulario de consulta incluyese un campo que permitiese introducir un precio máximo para los libros obtenidos en la respuesta de la consulta, entonces dicha capacidad sí se incluiría entre las positivas.

La razón por la que conocer estas capacidades es útil al mediador es que, siempre que sea posible, el mediador debe intentar pasar a la fuente la mayor cantidad de procesamiento que sea posible, reduciendo así la cantidad de procesamiento local, a la vez que la cantidad de información trasmitida por la red. En realidad se trata de un mecanismo de optimización: delegar a las fuentes la mayor parte de procesamiento posible para minimizar el transito de datos a través de la red y además descentralizar la carga de trabajo.

Por lo tanto, las *capacidades positivas* ayudan al mediador a generar planes de consulta *óptimos*.

Se han propuesto diversos lenguajes en la literatura para la representación de las capacidades de consulta positivas y negativas de las fuentes [PGH96][YLGU99]. Las principales ideas utilizadas en ellos se describen someramente en el siguiente apartado.

II.5.1.3.1. DESCRIPCIÓN DE LAS CAPACIDADES DE CONSULTA DE LAS FUENTES

La mayor parte de los sistemas mediadores construidos hasta el momento utilizan las consultas conjuntivas como lenguaje de consulta. El modelado de las *capacidades negativas* de las fuentes en un sistema de este tipo puede realizarse asignando a cada una de ellas un conjunto de *patrones de consulta*.

Si una fuente es modelada por una relación con `n` atributos, entonces un *patrón de consulta* es una cadena de caracteres de longitud `n`, compuesta por las letras `b` y `f`. El significado de la letra `b` (que proviene de la palabra inglesa 'bound' e indica obligatoriedad) en una determinada posición del patrón es que una consulta sobre la fuente debe obligatoriamente proporcionar un valor para el atributo situado en esa posición. El significado de la letra `f` (que proviene de la palabra inglesa 'free', que

indica opcionalidad) en una determinada posición del patrón es que las consultas pueden incluir o no un valor para el atributo en esa posición.

Ejemplo II.5.1.6: Patrones de Consulta

Supóngase una tienda electrónica de libros modelada mediante una relación `R={TITULO,AUTOR,PRECIO}`. La fuente ofrece un formulario de consulta que exige rellenar el campo correspondiente al atributo `TITULO` para que la consulta sea contestada.

Las capacidades de consulta *negativas* de `R` pueden expresarse como $R(TITULO,AUTOR,PRECIO)^{bff}$. Así, una consulta conjuntiva `q` que incluya al menos una submeta como `R(t,AUTOR,PRECIO)` será permitida por la fuente ya que proporciona un valor para el único atributo que tiene la letra `b` asignada, con lo cual el campo asociado en el formulario podrá ser rellenado y, así, las restricciones de la fuente satisfechas. Cualquier otra submeta incluida en `q` podrá siempre ser ejecutada por el mediador aplicando técnicas de post-procesado como las mencionadas en el ejemplo II.5.1.5.

□

Como puede verse, este sencillo método permite representar las *capacidades negativas* de las fuentes.

Para representar las *capacidades positivas* puede extenderse el esquema anterior, añadiendo la letra `u` (del término inglés 'unspecified', indicando que el atributo no puede ser especificado en las consultas sobre la fuente) y cambiando ligeramente el significado de la letra `f`. Ahora la letra `f` se asignará a aquellos atributos opcionales cuyos predicados de selección pueden ser ejecutados directamente por la fuente, mientras que la letra `u` se reserva a aquellos atributos opcionales cuyos predicados la fuente no puede ejecutar por sí misma (aunque el mediador sí pueda ejecutarlos en ocasiones mediante la aplicación de técnicas de post-procesado (ver ejemplo II.5.1.5)).

Una extensión adicional sobre este esquema, propuesta en [YLGU99], propone las siguientes letras adicionales para representar más adecuadamente el frecuente caso en el que está acotado el rango de valores que es posible utilizar en los predicados de selección con un cierto atributo:

- *c[s]*. Este indicador representa que el atributo debe ser especificado en la consulta (como en el caso de *b*), pero además el valor especificado tiene que ser forzosamente uno de los contenidos en el conjunto *s*.
- *o[s]*. Este indicador representa que el atributo puede o no ser especificado en la consulta (como en el caso de *f*), pero si se especifica, el valor tiene que ser forzosamente uno de los contenidos en el conjunto *s*.

Un ejemplo de aplicación para esta extensión lo podemos encontrar en cualquier formulario web que incluya algún campo de tipo lista de selección y que, por lo tanto, restringe el rango de valores utilizables en las consultas para ese atributo a una serie de valores predeterminados.

Es importante resaltar de nuevo que el mecanismo de *patrones de consulta* que se ha descrito en este apartado es sólo válido para mediadores que utilicen consultas conjuntivas como lenguaje de consulta y no es válido directamente para lenguajes de consulta que permitan diferentes operadores. Nótese que en las consultas conjuntivas se asume implícitamente el operador de igualdad para todos los predicados de selección, por lo cual si alguna fuente tiene capacidades de consulta que utilicen otros operadores no tiene sentido representarlas ya que, de todas maneras, esas consultas no podrían ser expresadas en dicho lenguaje. De hecho, se asume que el WHERE de cualquier consulta realizada sobre una relación de una fuente es del tipo:

```
(A0=a0) and (A1=a1) and ... (An=an)
```

dónde `A0,...,An` son atributos de la relación consultada y `a0...an` son los valores consultados. No es posible, por ejemplo un predicado WHERE del tipo:

```
(A0=a0) and (A1<a1) and (A1>a3) and (A2 contiene a2)
```

dónde se utilizan otros operadores distintos del de igualdad (en concreto, los operadores '<','>' y el operador `contiene` entre cadenas de caracteres) y además hay más de una condición de selección para el atributo *A1*.

Para el modelado de muchas fuentes, esto es claramente insuficiente, traduciéndose en el mejor de los casos en un desaprovechamiento de la capacidad de procesamiento local de las fuentes y, en el peor, en la obtención de resultados erróneos o la imposibilidad de realizar consultas que la fuente sí permite.

En [PGH96] se propone un mecanismo de representación de capacidades de consulta positivas y negativas significativamente más expresivo que el de los patrones de consulta y que sí soporta la utilización de lenguajes de consulta más expresivos que las consultas conjuntivas. Sin embargo, presenta también un inconveniente cuya importancia se detallará en el apartado II.5.1.3.3: no permite calcular las capacidades de consulta de las relaciones del esquema mediado.

Como se verá más adelante, el sistema mediador que se presenta en este trabajo aborda esta cuestión mediante un marco propio de representación de capacidades con un poder expresivo mayor que en [YLGU99], mediante la inclusión de la posibilidad de utilizar diferentes operadores (no sólo el de igualdad) y de tratar fuentes que permiten la ejecución de varias condiciones de selección simultáneas sobre el mismo atributo. También se verá que el sistema presentado en esta tesis doctoral evita el inconveniente apuntado para [PGH96], ya que proporciona también algoritmos para calcular las capacidades de consulta de las relaciones del esquema mediado.

II.5.1.3.2. REFORMULACIÓN TRATANDO LIMITACIONES EN LAS CAPACIDADES DE CONSULTAS

En los métodos de reformulación de consultas mostrados en los apartados II.5.1.1 y II.5.1.2 no se tuvieron en cuenta las posibles limitaciones en las capacidades de consulta de las fuentes. En este apartado se resumen las principales ideas a tener en cuenta para extender dichos métodos de manera que puedan soportar adecuadamente dicho supuesto.

La principal idea a tener en cuenta es que un plan de ejecución factible en un mediador no puede contener ninguna subconsulta para una determinada fuente que no satisfaga las restricciones impuestas por la misma (expresadas a través de sus capacidades negativas).

Por lo tanto, en el enfoque GAV es necesario extender el generador de planes de consulta para que elimine aquellos planes que no cumplan esta característica.

En el enfoque LAV, las extensiones necesarias son más complicadas ya que estas condiciones complican aún más la acotación de la búsqueda dentro del espacio de posibles soluciones que involucra cada proceso de reformulación (véase el apartado II.5.1.2.1 para recordar la descripción de este proceso).

Existen algunos resultados positivos a este respecto. En [KWO96] se demuestra que, aunque con un límite algo mayor, el problema de encontrar reformulaciones equivalentes de una consulta sigue siendo NP-completo en presencia de limitaciones en las capacidades de consulta expresadas mediante patrones de consulta simples (usando sólo las letras `b` y `f`). Sin embargo, en el mismo trabajo se demuestra también que el problema de encontrar reescrituras que estén contenidas de manera maximal en la consulta (véase el apartado II.5.1.2.1 para recordar la descripción de este concepto), no es NP-completo sino que puede llevar incluso a tiempos infinitos de ejecución en ciertos casos. Afortunadamente, en [DGL00] se demuestra que mediante el uso de planes de consulta recursivos, es posible también tratar de manera adecuada la mayor parte de estos casos conflictivos, si bien este resultado es válido sólo para consultas conjuntivas puras, sin que sean válidos en las consultas ni siquiera predicados de comparación (que sí pueden ser utilizados en algunos algoritmos de reformulación previos).

Como conclusión, puede decirse que la introducción de limitaciones en las capacidades de consulta de las fuentes introduce complicaciones notables en los algoritmos necesarios para la reformulación de consultas en sistemas LAV. Sin embargo, existen soluciones que permiten tratar el problema en tiempos razonables para la mayoría de casos siempre que el lenguaje de consultas sea simple (consultas conjuntivas) y la descripción de las capacidades siga un esquema sencillo como el de patrones de consulta.

II.5.1.3.3. SISTEMAS DE CÁLCULO DE LAS CAPACIDADES DE CONSULTA DE LAS RELACIONES DEL ESQUEMA MEDIADO

La mayoría de los sistemas mediadores construidos hasta la fecha, como por ejemplo TSIMMIS[GMPQ95], Garlic[HKWY97], Information Manifold[LRO96] o DISCO[TRV98], tienen en cuenta las capacidades de consulta de las fuentes utilizando mecanismos como los descritos en apartados anteriores. Todos ellos son capaces también de descartar los planes de ejecución de consultas que no son factibles de acuerdo a dichas capacidades.

Sin embargo, ninguno de estos sistemas calcula las capacidades de consulta de las relaciones del esquema mediado basándose en las capacidades de las relaciones de las fuentes.

Como ejemplo de este problema, considérese la situación descrita en el ejemplo II.5.1.1 (presente en el apartado II.5.1.1), y supóngase además que las fuentes A, B y C presentan limitaciones en sus capacidades de consulta expresadas de acuerdo a la técnica de *patrones de consulta* descrita en el apartado II.5.1.3.1. Conociendo las capacidades de consulta de A, B y C, así como la expresión que define R como $R(X,Y,Z,K)=(A \cup B) \infty_Y C$, ¿cuáles son los patrones de consulta que expresan las capacidades de consulta de R?

Nótese que si no se calculan las capacidades de consulta de R, el sistema mediador no puede saber *a priori* qué consultas sobre R podrán ser respondidas y cuáles no: lo único que puede hacer el mediador al recibir una consulta es tratar de ejecutarla y, si al generar todos los posibles planes de ejecución para ella descubre que no existe ningún plan factible dadas las capacidades de consulta de las fuentes, informar al usuario *a posteriori* de que no podrá ser contestada. Sin embargo, no hay ninguna manera de que el usuario pueda saber *a priori* si una determinada consulta podrá ser respondida o no por el mediador.

Esto causa dos importantes inconvenientes:

- El usuario está forzado a realizar las consultas mediante un proceso de prueba y error, enviando consultas que son rechazadas, hasta que finalmente el mediador es capaz de responder a alguna consulta. Esto es claramente inconveniente.
- No es posible que unos mediadores actúen como fuentes de otros mediadores, ya que los mediadores de nivel superior precisan las capacidades de las relaciones del esquema mediado de nivel inferior para determinar si los planes de consulta generados son factibles o no. Esta característica es, sin embargo, altamente deseable ya que hace posible la construcción de arquitecturas de mediación en varias capas, que pueden facilitar mucho el desarrollo incremental de proyectos complejos.

Por tanto, se concluye que es altamente deseable poder calcular a priori las capacidades de consulta de las relaciones del esquema mediado.

Sin embargo, en general, esta tarea es complicada debido a una serie de factores:

- La variedad de capacidades de consulta de las fuentes
- La multitud de técnicas que los mediadores pueden utilizar para procesar las consultas (filtros, diversas técnicas de ejecución de joins, etc.).
- Las técnicas especiales de post-procesado (ver apartado II.5.1.5) que pueden aplicar los mediadores para extender las capacidades de consulta de las fuentes
- El hecho de que ciertas consultas podrán o no ser respondidas por el mediador en función de los contenidos concretos que tenga una fuente en un determinado momento. En estos casos, el mediador debe definir sus capacidades con una "banda inferior" que especifica qué consultas se podrán responder siempre, y una "banda superior" indicando qué consultas no se podrán responder nunca. Las consultas entre ambas bandas podrían ser respondidas dependiendo de los contenidos de las fuentes en el momento de la ejecución. A estos mediadores que tratan con bandas inferiores y superiores se les conoce como *mediadores dinámicos*.

A pesar de su importancia, este problema apenas ha sido abordado en la literatura. [YLGU99] es una excepción. En dicho trabajo se presentan algoritmos para calcular las capacidades de consulta de las relaciones del esquema mediado para mediadores que utilicen el enfoque GAV, partiendo de las de las fuentes. Sin embargo, el marco de representación de capacidades de consulta utilizado no es, tal y como ya se detalló en el apartado II.5.1.3.1, suficientemente expresivo para tratar con toda la complejidad encontrada en casos reales.

Tal y como ya se ha mencionado, y como se tratará en detalle en la sección III.2, el sistema mediador que se presenta en este trabajo aborda esta cuestión mediante un marco propio de representación de capacidades con un poder expresivo mayor que en [YLGU99], y que proporciona además un algoritmo para calcular las capacidades de consulta de las relaciones del esquema mediado (ver apartado III.2.4.1).

II.5.2. EJECUCIÓN Y OPTIMIZACIÓN DE CONSULTAS

II.5.2.1. CONSTRUCCIÓN DE MOTORES DE EJECUCIÓN

El problema de construir motores de ejecución de consultas ajustados a las características particulares de los mediadores ha recibido comparativamente poca atención hasta el momento, si bien empieza a mostrar ya señales de una actividad mayor.

Los problemas principales para la construcción de estos sistemas provienen de la autonomía de los datos de las fuentes y de lo impredecible del rendimiento de las redes.

En particular, cuando se accede a las fuentes web se pueden experimentar retardos iniciales antes de que los datos sean transmitidos, y cuando ya se ha iniciado su transmisión la llegada al destino puede ser a ráfagas.

Los motores de ejecución de consultas de mediadores deberían disponer de características como:

- Ser capaces de funcionar basándose en procesos de optimización dinámicos, en los cuales las asignaciones de costes pueden variar de forma dinámica incluso durante el transcurso de una consulta (e.g una fuente congestionada), y la utilización de planes alternativos óptimos y/o redundantes. El mediador debe ser capaz de "reoptimizar" si la información recolectada durante la ejecución de una consulta sugiere que el plan escogido no era realmente el óptimo.
- Ser capaces de funcionar de manera asíncrona. Muchas de las fuentes de un mediador entregarán sus datos en última instancia a través de algún tipo de "corriente" de datos ('stream'). El sistema debería ser capaz de tratar los datos que ya han llegado al sistema antes de que la respuesta de la fuente esté completa. Esto permite, que en los dominios de aplicación donde esto sea útil, el sistema pueda devolver los primeros resultados a una consulta tan pronto como estén disponibles, sin necesidad de esperar a que la consulta esté totalmente procesada. En sistemas con altas latencias de red, esta funcionalidad es muy interesante.
- En el caso de que en un instante de tiempo determinado falle un plan y no exista ninguno alternativo, deben permitir reintentos futuros del plan que ha fallado, siempre que el origen del fallo así lo aconseje (por ejemplo, si la ejecución de un plan ha fallado por sobrecarga de la red o por una caída del servidor).
- Dependiendo del dominio de aplicación concreto en el que se emita una consulta, el motor de ejecución y optimización debería poder ser configurado para adoptar un comportamiento u otro. Para algunas aplicaciones, resultados incompletos o incluso sólo aproximados, pueden ser aceptables pero a cambio se requerirán tiempos de respuesta menores. En otros contextos, los tiempos de respuesta deberán estar supeditados a la total corrección y completitud de los datos. El motor de ejecución y optimización debe poder variar su funcionamiento dinámicamente para cada caso.
- Para optimizar los tiempos de respuesta, deben usarse sistemas de cache. Esto debe ser conciliado con la importancia de garantizar que las respuestas devueltas estén actualizadas. En un contexto en el que las fuentes son fuertemente autónomas, minimizar los problemas derivados de la falta de actualidad de los datos es especialmente difícil.

Los primeros sistemas en abordar parte de estos requerimientos (fundamentalmente la capacidad de rehacer los planes de consulta si hay retardos iniciales de acceso a las fuentes) han sido [AFT98][UFA98][STD00] (sistema Niagara) y [ILWF00] (sistema Tukwila).

II.5.2.2. OPTIMIZACIÓN DE CONSULTAS

En cuanto a lo que se refiere a la optimización de consultas, nuevamente el entorno habitual de los mediadores plantea problemas adicionales a los encontrados en bases de datos convencionales. En [HS01] puede encontrarse una panorámica del estado actual del campo.

La estimación de costes en bases de datos relacionales utiliza estadísticas de la base de datos y fórmulas para computar el coste de cada operador en cada plan (ver [Cha00] para una buena panorámica del estado actual del arte).

Son técnicas no directamente aplicables a los sistemas mediadores debido a que:

- Las fuentes no suelen ofrecer información estadística.
- Las fórmulas de coste para cada operador varían dependiendo de:
  - La implementación del envoltorio de la fuente.
  - La implementación de la fuente remota.
- Los costes de comunicaciones no se determinan fácilmente y pueden variar.

Los primeros trabajos de optimización de consultas en sistemas de integración de datos en el web se centraron en el problema de seleccionar un conjunto mínimo de fuentes a las que acceder, y en determinar la consulta mínima a cada una de estas fuentes.

En cuanto al primer objetivo de minimizar el número de fuentes accedidas, a menudo ocurre, sin embargo, que estas técnicas son de escasa ayuda pues el conjunto de fuentes de las que se dispone rara vez es redundante y la estrategia para conseguir el mínimo número de accesos a fuentes es evidente.

Por lo que respecta al objetivo de enviar la consulta mínima a cada fuente, el foco en la investigación realizada hasta el momento ha estado en aprovechar lo más posible las capacidades positivas de las fuentes de manera que pueda delegarse en ellas el máximo procesamiento posible. La idea básica es disminuir lo más posible la cantidad de datos que deben ser transferidos entre las fuentes y el mediador. Por ejemplo, si una fuente realiza ella misma una selección de sus tuplas por el valor de un determinado campo y devuelve el resultado, entonces la cantidad de datos a transmitir por la red es mucho menor que si la fuente enviase todas sus tuplas y fuese el mediador quien aplicase el criterio de selección.

Esta técnica de optimización no es muy diferente de la aplicada en los sistemas de bases de datos convencionales que consiste en propagar hacia abajo todo lo posible las selecciones y proyecciones en el árbol algebraico de una consulta. La diferencia principal es que el mediador, al propagar las selecciones y proyecciones hacia abajo, debe asegurarse de que las capacidades de consulta de las fuentes las permiten.

Ejemplo II.5.2.1: Optimización en álgebra relacional

Supóngase un sistema mediador con tres fuentes A(X,Y,Z), B(X,Y,Z) y C(Y,K). Supóngase que una de las relaciones del esquema mediado viene definida como R(X,Y,Z,K) = (A∪B)∞yC.

Supóngase ahora que el mediador recibe la consulta expresada en álgebra relacional:

$\sigma_{(x=x0 \text{ and } k=k0)}(R)$

que en lenguaje sql se escribiría:

```
select * from r where x=x0 and k=k0
```

El mediador desplegará las vistas para obtener la reformulación de la consulta mostrada en (1). Puede optimizarse el proceso propagando las selecciones a las fuentes, tal y como se muestra en (2). Sin embargo, esta propagación sólo podrá realizarse si las capacidades de consulta de A, B y C lo permiten:

1. $\sigma_{(x=x0 \text{ and } k=k0)}(R) = \sigma_{(x=x0 \text{ and } k=k0)}((A \cup B) \infty_y C))$
2. $= (\sigma_{x=x0}(A) \cup \sigma_{x=x0}(B)) \infty_y \sigma_{k=k0}(C)$

□

Otro concepto que ha sido explotado para ayudar en las tareas de optimización de consultas es el de "completitud de las fuentes". En general, las fuentes que nos encontramos en el WWW no son necesariamente completas en relación al dominio que tratan, aunque pueden existir algunas excepciones (como por ejemplo la base de datos en línea del ISBN con respecto a los libros publicados en España). El conocimiento sobre el grado de completitud de una fuente web puede ayudar al sistema de integración en diferentes aspectos. Por ejemplo si se obtiene una respuesta negativa de una fuente completa, entonces el sistema de integración de datos podría cortar el acceso a otras fuentes.

Para el caso particular de la optimización de "consultas de fusión" (que son una clase especial de consultas de integración que se centran en la recuperación de varios atributos de un objeto dado desde múltiples fuentes), un trabajo interesante es[YPAG98]. El enfoque utilizado es atractivo, pero el ámbito de aplicación está restringido a este tipo especial de aplicaciones.

A pesar de que estas técnicas comentadas hasta ahora pueden ser útiles, están aún lejos de proporcionar un enfoque sistemático para la optimización, que permita a las fuentes exportar su información de costes (que puede ser heterogénea e incompleta) y, partiendo de ella, obtener la información de costes para las relaciones del esquema global mantenido por el mediador. En definitiva, se precisa un enfoque menos

heurístico y más sistemático, similar en cuanto a funcionalidad al utilizado en las bases de datos de tradicionales.

Recientemente han aparecido varios trabajos que abordan este problema desde diferentes puntos de vista y que, si bien aún no se ha alcanzado una solución general plenamente satisfactoria, suponen un importante paso adelante en el campo que nos ocupa. A continuación se repasan los principales.

En [NGT98], los autores defienden un modelo genérico de costes mantenido por el sistema mediador, pero permitiendo que los envoltorios de las fuentes aporten información específica que "reescriba" partes de ese modelo general.
De esta manera, el envoltorio especifica la información de la que disponga, y, la que no conozca, será rellenada por el mediador con información por defecto. Así, el sistema puede tratar los casos en que se disponga de información incompleta sobre las fuentes.

Tal y como está concebido, este concepto es muy adecuado cuando la información es muy homegénea entre sí.

Desgraciadamente, en este trabajo, no se tiene en cuenta el coste menos medible de todos, que es el de comunicaciones, lo cual lo hace de escasa utilidad en un sistema mediador real, pues la estimación ofrecida por el modelo de costes "por defecto" del mediador puede no sólo ser inadecuada, sino que incluso puede empeorar los tiempos de respuesta.

Otro problema del modelo de [NGT98] es que el modelo de costes genérico del mediador contiene sólo los siguientes tres parámetros temporales de medida:

- TimeFirst: tiempo de obtención de la primera tupla.
- TimeNext: tiempo entre obtención de tuplas subsiguientes.
- TotalTime: tiempo de obtención de todas las tuplas.

Aunque estos tres parámetros son suficientes en un sistema de bases de datos convencional, los modernos sistemas mediadores tienen que enfrentarse con fuentes cuyas características son difíciles de modelar con fidelidad con estos parámetros.

Por ejemplo, en fuentes web (sin duda, uno de los principales tipos de fuentes con los que tratan los mediadores), es habitual que la respuesta a las consultas se produzca dividida en diferentes páginas. Así, si una consulta devuelve 200 resultados, es muy posible que estos aparezcan distribuidos en 10 páginas web conteniendo 20 resultados cada una.

Esta situación, en la que la respuesta a una consulta llega al mediador dividida en segmentos, no puede ser representada sólo con los tres parámetros anteriores (nada tiene que ver el tiempo entre la tupla 1 y 2 con el tiempo entre la tupla 20 y 21, donde se produce un cambio de segmento). Para tratar esos casos, se requiere al menos el

tiempo de obtención de la primera tupla, el tiempo de obtención de la siguiente tupla dentro de un mismo segmento y el tiempo de obtención del siguiente segmento.

Otro trabajo interesante es [ROH00]. Si bien este trabajo centra su atención en el ámbito de las bases de datos federadas, algunas de sus ideas son aplicables también para los sistemas mediadores.

Al igual que en el caso anterior, los envoltorios pueden proporcionar cierta información de costes que el mediador utilizará. Para los casos en los que los envoltorios no proporcionen información, el mediador utilizará formulas generales.

Para casos de optimización en sistemas convencionales, utiliza los siguientes parámetros:

- Coste total: coste en segundos para ejecutar un determinado operador.
- Coste de re-ejecución: coste de ejecutar el plan una segunda vez (para tener en cuenta mecanismos de cache).
- Cardinalidad. Número de tuplas estimado de una relación.

Para el caso de bases de datos heterogéneas (similar al de los mediadores), propone unas fórmulas por defecto para fuentes simples, que permiten calcular el valor de los parámetros anteriores:

```
1) Coste total = coste de comienzo + coste de avance X
(cardinalidad/tamaño de bloque)

2) Coste de reejecución = coste total - coste de comienzo.

3) Cardinalidad = Productorio_i (cardinalidad base (i) X selectividad
de predicados)
```

El "coste de comienzo" es análogo al "TimeFirst" de [NGT98]. Nótese como en este caso si se tiene en cuenta el hecho de que los datos pueden venir divididos en segmentos (bloques). El "coste de avance " se refiere al coste de pasar de un segmento a otro dentro de la lista de resultados.

Otro tipo de opción es el de "calibración"[WKM00]. En este enfoque, utilizado fundamentalmente para sistemas de bases de datos tradicionales, existe un único modelo de costes genérico, que los envoltorios no pueden "reescribir" en parte con su propia información. Los métodos de calibración permiten estimar los principales coeficientes de dicho modelo de costes único, para ajustarse a las peculiaridades del sistema concreto a optimizar.

Los enfoques basados en calibración no parecen muy interesantes desde el punto de vista de los sistemas mediadores o de las bases de datos heterogéneas, ya que al basarse en un único modelo de costes que los envoltorios no pueden ni siquiera

sobrescribir en parte, sólo ofrecerá resultados adecuados en los casos en que haya una gran homogeneidad entre las fuentes.

Quizás la solución más interesante propuesta hasta el momento para sistemas mediadores sea [ACPS96] utilizada dentro del sistema Hermes.

En ella, el modelo de costes se evalúa a partir de información histórica. El sistema va almacenando la información de costes de las consultas efectuadas sobre cada fuente. Partiendo de las estadísticas obtenidas sobre las consultas reales en cada fuente, se realiza la estimación del coste de los planes de ejecución para las consultas nuevas.

Este enfoque tiene las siguientes características:

- Es muy útil para aquellas fuentes utilizadas uniformemente. Una fuente es utilizada uniformemente si las consultas que recibe son similares entre sí. De esta manera, los datos históricos tendrán validez. Si una fuente es consultada de maneras muy diferentes, los datos recogidos para consultas previas pueden ser una mala guía para nuevas consultas.
- A diferencia de los casos anteriores, no le afecta el que las fuentes sean muy heterogéneas entre sí: lo que importa es que cada fuente individual sea consultada siempre de una manera similar, pero entre fuentes distintas puede haber tantas diferencias como se quiera sin que la eficacia del sistema se vea afectada.
- La información de costes sobre las fuentes se obtiene por la misma operación del sistema, sin que sea necesario disponer de ningún tipo de conocimiento sobre el funcionamiento interno de las fuentes. En los contextos en los que suelen funcionar los sistemas mediadores, donde las fuentes tienen una amplia autonomía (piénsese por ejemplo en el caso de las fuentes web), esta característica es una importante ventaja.
- Es poco útil para fuentes poco utilizadas o cuando se consulta una fuente de una manera poco frecuente (ya que, en estos casos, habrá pocos datos históricos o estos serán poco relevantes). Lo mismo ocurre en los primeros momentos de utilización del sistema.

El requerimiento de que las fuentes sean utilizadas uniformemente es, para un sistema mediador, menos fuerte que el que las fuentes tengan que ser homogéneas entre sí. Muchas de las fuentes típicas en un sistema mediador, tales como las fuentes web o las basadas en algún tipo de fichero estructurado, ofrecen interfaces de consulta bastante limitadas, por lo que, de manera casi forzosa, las consultas sobre cada una de esas fuentes serán muy parecidas entre sí. Por el contrario, la idea de que las fuentes tengan que ser homogéneas, va contra la misma idea de mediador.

Por lo tanto, este enfoque es considerablemente atractivo para sistemas mediadores.

De todas maneras, un sistema mediador que aspire a la generalidad, debe proporcionar además alguna funcionalidad que permita al sistema ofrecer un comportamiento mejor para el caso de fuentes poco utilizadas o consultadas de manera

no uniforme. Una combinación de este método con algunas de las ideas de los trabajos comentados anteriormente, en la que se utilizase un modelo general de costes cuando no hubiese información estadística relevante para una fuente, podría ser la solución.

Por otra parte, hasta ahora sólo se han comentado las diferentes opciones que tiene el mediador de obtener información numérica o estadística sobre las fuentes a partir de los envoltorios.

La segunda parte de una optimización basada en costes es la obtención del coste de un plan de ejecución en concreto, de manera que sea posible escoger, para una consulta determinada, aquel plan de entre los posibles que presente un coste menor.

En [WKM00], cada operador (ya sea binario o unario) tiene asociado una fórmula que depende de una serie de parámetros temporales y espaciales (p.e. número medio de tuplas devueltas en una petición).

Para obtener el coste de un plan de consultas dado, este sistema utiliza el árbol de procesamiento del plan de consulta.

Con la información de costes de que dispone para las fuentes, se realiza un proceso de abajo a arriba, que comienza por las operaciones binarias/unarias que afectan directamente a las hojas (fuentes), y va propagando los costes en sentido ascendente por el árbol del plan de ejecución, en función de las fórmulas de costes definidas para los operadores que va encontrando en los nodos del árbol. Una vez que se ha llegado a la raíz del árbol, se ha obtenido ya el coste del plan.

Por lo tanto, el proceso a realizar por el optimizador consistiría en utilizar este método para calcular el coste de todos los posibles planes de ejecución y, entonces, escoger el plan de menor coste.

[WKM00] también define fórmulas para los principales operadores unarios y binarios, así como para los principales modos de acceso a los datos. Por brevedad, no se exponen aquí las fórmulas concretas utilizadas, pero sí se listan todos los operadores tratados.

Por un lado, para operadores unarios, se considera el operador *acceso* (o '*scan*') que permite acceder a una tupla o conjunto de tuplas partiendo de un valor para un atributo. Se consideran los dos siguientes modos de acceso:

- Acceso secuencial. Consiste en recorrer todas las tuplas de la relación hasta encontrar la buscada
- Acceso indexado. En este caso el acceso puede realizarse a través de un índice.

Para cada caso, se obtiene una fórmula de coste, que depende de los parámetros: mínimo valor para el atributo, máximo valor para el mismo y número de valores distintos.

En el caso de los operadores binarios, se definen los siguientes:

- Join indexado.
- Join anidado ('nested loops'): Una tabla se selecciona como tabla exterior, y otra como interior. La base de datos accederá a la tabla exterior fila a fila, y, para cada una, buscará en la interior para intentar encontrar filas que cumplan la condición del join. Este tipo de suele ser elegido por el optimizador de bases de datos si una de las tablas es pequeña y la otra tiene un índice sobre la columna que une las tablas. Su complejidad algorítmica es n^2. (NOTA: como se verá con mayor claridad en secciones posteriores de este documento, este tipo de join no sólo se utiliza por motivos de optimización sino que, además, en ocasiones puede ser el único permitido por las capacidades de consulta de las fuentes).
- Join ordenado: se ordenan las dos tablas antes, y después se unen ordenadamente fila a fila. Su complejidad algorítmica es n*log (n)).

Cuando existe un índice, se elige el join indexado. Si no, se escoge el mejor de los otros dos.

Una dificultad que se encuentra con estrategias de este tipo, que dependen de si la fuente está indexada o no, es que en fuentes autónomas es posible que no se disponga de esa información.

En [HKWY97] se aplica una idea similar a [WKM00]. El generador de planes se encarga de la construcción de un conjunto de planes de consulta posibles para después, mediante un proceso de abajo a arriba a través de su árbol, obtener el coste total de cada plan, y elegir uno. Este plan es después traducido a un formato ejecutable o interpretable.

En este caso, el proceso de optimización interviene ya durante la creación de planes ya que, según estos se van creando, el sistema es capaz de cortar la generación de aquellos que sean un subconjunto de otro que además presente un menor coste. De esta manera, parte del proceso de optimización se realiza ya durante el proceso de generación de planes, y la propagación de costes de abajo hacia arriba para calcular el coste del plan sólo es necesario realizarla con los planes "finalistas" obtenidos durante el proceso de generación. Así, el coste del proceso de optimización es menor.

Como conclusión y resumen sobre el tema de la optimización, se ofrecen las siguientes ideas:

- Quizás la mejor manera de recolectar información de costes sobre las fuentes en un sistema mediador es mediante el mantenimiento de estadísticas de costes sobre las consultas emitidas anteriormente sobre el sistema. Este enfoque permite recoger información valiosa sobre las fuentes independientemente de su grado de autonomía y no requiere para funcionar bien que la información de las fuentes sea homogénea, si bien sí requiere cierto grado de uniformidad en las consultas recibidas por cada fuente para que los históricos ofrezcan información útil.

- Para tratar los casos en que no haya información de costes disponible sobre las fuentes o ésta no sea útil para una consulta concreta, el mediador debe ofrecer algún tipo de modelo general que permita adoptar estrategias razonables.
- Una vez obtenida la información de costes sobre las fuentes, la obtención del coste total de un plan de consulta puede realizarse propagando los costes de abajo a arriba a través del árbol del plan, utilizando fórmulas de costes para los operadores que pueden aparecer en dichos árboles. De esta manera, puede escogerse el plan con menor coste.
- Como ya se ha comentado, sería conveniente la realización de optimizadores y motores de ejecución que fuesen capaces de reaccionar dinámicamente, incluso sobre la marcha en la ejecución de una consulta, a cambios inesperados en los costes de las fuentes. Por ejemplo, si una fuente está congestionada en un momento dado, los costes reales de consulta sobre esa fuente serán muy superiores a los teóricos y sería conveniente que el sistema pudiese reaccionar dinámicamente ante ello. Este es todavía un tema abierto de investigación. Los primeros sistemas en abordar este tipo de cuestiones son Niagara[STD00] y Tukwila[ILWF00].

II.5.3. TRATANDO CON LAS HETEROGENEIDADES DE REPRESENTACIÓN

Uno de los problemas más complicados cuando se realizan consultas sobre un conjunto de fuentes, es reconocer que dos objetos mencionados en dos fuentes diferentes, se refieren a la misma entidad. Este problema deriva de que cada fuente puede emplear sus propias taxonomías, así como sus convenciones de nombramiento y escritura.

Ejemplo II.5.3.1: Heterogeneidades de representación

Supóngase un mediador que integra información procedente de dos bases de datos de referencias bibliográficas. Ambas fuentes pueden ser consultadas por el título de la referencia, por su autor o por la materia a la que pertenecen. Sin embargo, las taxonomías de materias difieren en cada fuente. En una de ellas las materias posibles se agrupan en dos categorías fundamentales: 'artículos sobre sistemas de integración de datos' y 'artículos sobre bases de datos convencionales'. La primera categoría se subdivide a su vez en otras dos: 'artículos sobre bases de datos federadas' y 'artículos sobre sistemas mediadores'.

La segunda categoría de primer nivel se subdivide a su vez en otras tres: 'artículos sobre BD relacionales', 'artículos sobre bases de datos orientadas a objetos' y 'artículos sobre otros tipos de bases de datos convencionales'.

En la segunda fuente, sin embargo, la taxonomía de materias es muy diferente. Existen sólo dos categorías: 'artículos sobre bases de datos relacionales' y 'otros artículos relacionados con bases de datos'.

Para el mediador se plantea el problema, en primer lugar, de qué taxonomía de materias ofrecer en el esquema global, que sea coherente. Además, tendrá que reformular las consultas recibidas sobre la taxonomía del sistema global para cada una de las fuentes particulares.

Además de la heterogeneidad de taxonomías, existen también divergencias en los formatos de representación: una de las fuentes representa los nombres de los autores con el formato '`Apellido1 Apellido 2, Nombre`' mientras que la otra utiliza el formato '`Nombre Apellido1`'. Cada una de las fuentes espera que en las consultas por autor, el nombre del mismo aparezca representado en su formato particular. Además, lógicamente, los resultados devueltos por cada fuente irán también en su formato.

□

Pueden ocurrir casos aún más sutiles. Supóngase, por ejemplo, que dos fuentes utilizan claves compuestas por diferentes campos. Por ejemplo, supóngase dos fuentes que contengan referencias sobre películas. Una fuente puede identificar sus tuplas mediante su título y el nombre de su director y no ofrecer un campo con información sobre los intérpretes de la película. Otra fuente, sin embargo, puede hacerlo por el título y el nombre de su actor o actriz principal y no proporcionar el nombre del director. En estas condiciones, identificar como iguales dos tuplas procedentes de estas dos fuentes tendría que hacerse exclusivamente en base al campo título. Sin embargo, incluso aunque el formato de representación sea el mismo, esto podría llevar a errores ya que existen películas diferentes con el mismo título.

En los casos más simples, algún campo o conjunto de campos de las tuplas constituirán una clave con el mismo formato para todas las fuentes, permitiendo así identificar de manera trivial las tuplas que se refieran al mismo objeto. Por ejemplo, si las tuplas recogidas de las fuentes se refieren a libros, su campo ISBN podría ser utilizado para identificarlos de manera unívoca.

Cuando esto no es posible, la mayoría de los sistemas, a falta de métodos generales, se enfrentan a este problema usando heurísticas o información semántica específica del dominio concreto de aplicación, que si no todos, solucione al menos la mayor parte de los problemas. Este es el enfoque adoptado en [FHM94], donde se propone un mecanismo para describir metainformación de un determinado dominio de aplicación. La información proveniente de las fuentes trata de unificarse de acuerdo al dominio.

En [Coh98] se propone un novedoso método que utiliza técnicas de la disciplina de *Recuperación de Información* para proporcionar una identificación automática de tuplas de diferentes fuentes que representen al mismo objeto. Por ejemplo, un sistema de este tipo podría detectar automáticamente que los valores '`Javier Marías`' y '`Marías, Javier`' se refieren realmente a la misma entidad del mundo real.

Sin embargo, este enfoque funciona únicamente con casos simples y para campos textuales. Además no trata el aspecto de la heterogeneidad de taxonomías.

Por ello, quizás la solución más razonable en este momento para tratar este tipo de problemas es que el sistema mediador proporcione al administrador un mecanismo de definición de reglas que sea lo suficientemente flexible y expresivo para permitirle especificar las diferencias en taxonomías y formatos de representación entre fuentes. De esta manera, el sistema mediador, apoyándose en estas reglas, puede posteriormente manejar las heterogeneidades de manera transparente. Para que el mecanismo de reglas no se quede corto y pueda realmente tratar cualquier caso, es importante que el mecanismo en sí sea extensible. Este enfoque tiene el inconveniente de que las reglas de transformación deben ser creadas y mantenidas manualmente.

II.5.4. CONSTRUCCIÓN DE PROGRAMAS "ENVOLTORIO"

El papel de los programas "envoltorio" es abstraer las particularidades de las fuentes, proporcionando a las capas superiores del mediador una visión estructurada o semiestructurada de las mismas.

Más concretamente, podemos distinguir las siguientes cinco tareas para un programa envoltorio:

1. Representación de las capacidades de consulta de una fuente. Tal y como se comentó en apartados anteriores, las capacidades de consulta de una fuente pueden estar limitadas. El envoltorio debe ser capaz de representar dichas capacidades y de saber si una determinada consulta emitida sobre la fuente en el lenguaje general de consultas del mediador puede ser respondida o no por ésta.
2. Representación de los formatos de salida de las consultas. El envoltorio debe representar el formato en el que devolverá al mediador los resultados de las consultas efectuadas sobre la fuente. Esto incluye los atributos disponibles en la fuente, sus tipos de datos, posibles restricciones sobre los contenidos, etc.
3. Obtener y exportar información de costes. Para propósitos de optimización, es importante disponer de información de costes sobre las consultas efectuadas sobre las fuentes. Las labores de los envoltorios pueden incluir el obtener esta información (quizás por algún proceso estadístico sobre el comportamiento de la fuente en consultas previas, tal y como se vio en el apartado II.5.2.2).
4. Traducir las consultas recibidas del mediador (y expresadas, por lo tanto, en el lenguaje de consultas del mismo) al lenguaje de consultas de la fuente y ejecutar dicha consulta sobre la misma. Esto puede involucrar un proceso complejo que permita realizar sobre la fuente las operaciones precisas para obtener un documento o conjunto de documentos que contengan las respuestas a la consulta efectuada. Por ejemplo, en fuentes web, puede involucrar la realización de un proceso de navegación automática a través del sitio web y el rellenado automático de uno o más formularios HTML.

5. Del documento o conjunto de documentos obtenidos en la tarea 4), extraer las tuplas que constituyen la respuesta a la consulta. En muchas fuentes (e.g. fuentes web) las tuplas no aparecerán claramente identificadas, al igual que tampoco lo estarán los atributos que las constituyen. Los documentos que contienen las tuplas a extraer pueden además estar organizados en forma de árbol o grafo y el envoltorio puede tener que ser capaz de navegar a través de ellos. Por ejemplo, en una fuente web, las tuplas a extraer pueden estar distribuidas en diferentes documentos accesibles entre sí a través de hiperenlaces.

Debido a su crucial papel dentro de la arquitectura de los mediadores, para que los sistemas basados en estos puedan utilizar un número medio-alto de fuentes, es fundamental que los programas envoltorio puedan ser creados y mantenidos con el menor esfuerzo posible. Por ello, se ha invertido un importante esfuerzo investigador en desarrollar técnicas que permitan la generación automática o semiautomática de programas envoltorio.

Sin embargo, debido a que las técnicas de creación de envoltorios pueden diferir en gran medida en función del tipo de fuente sobre la que actúen, las técnicas de generación automática o semiautomática desarrolladas, a menudo se centran en un tipo concreto de fuentes. En realidad, las tareas 1), 2) y 3) de los programas envoltorios son susceptibles de ser realizadas de una manera uniforme, independientemente del tipo de fuente sobre la que actúe el envoltorio. Sin embargo, esto es prácticamente imposible para las tareas 4) y 5), fuertemente dependientes de la naturaleza de la fuente a tratar.

Las tareas 1) y 2) fueron abordadas por primera vez en [HGN98], dentro del sistema TSIMMIS, de la Universidad de Stanford e IBM. El enfoque adoptado para la tarea 1) consiste en representar las capacidades de consulta y respuesta de una fuente mediante un lenguaje declarativo basado en plantillas o patrones de consulta que representan aquellas consultas que pueden ser respondidas por la fuente. De esta manera, el creador del envoltorio puede limitarse a definir las capacidades de consulta en este lenguaje declarativo y el sistema será el encargado de, dada una consulta en el lenguaje del mediador, determinar si puede ser o no resuelta por la fuente.

Cuando se recibe una consulta por parte del mediador, el sistema contrasta la consulta recibida con los patrones de consulta disponibles en la fuente y determina si es o no contestable por la misma. Si es así, las tareas 4 y 5 pueden comenzar, para realizar la consulta y extraer los resultados de la misma (aunque estas tareas no son tratadas en el sistema mencionado, dónde deben ser programadas enteramente por el creador del envoltorio). Si no es así, se informa al mediador de la incapacidad de realizar la consulta.

En cuanto a la tarea 2), el enfoque utilizado es el mismo, pero aún más simple. El administrador puede utilizar un lenguaje para definir declarativamente los atributos devueltos por las consultas, sus tipos de datos y posibles restricciones sobre los mismos.

Prácticamente todos los sistemas mediadores posteriores han utilizado un enfoque similar a éste para la automatización de las tareas 1) y 2) de un envoltorio, si bien los lenguajes de declaración de capacidades y formatos de salida pueden diferir bastante y tener una capacidad expresiva bastante diferente.

En cuanto a la tarea 3), trabajos como [GRV98] y [ACPS96] (dentro del sistema HERMES) hacen que los envoltorios recolecten estadísticas de costes sobre las consultas realizadas sobre las fuentes. De esta manera, puede inferirse información de costes que sea útil en sucesivas consultas. Otros sistemas [NGT98] se centran en definir interfaces adecuadas entre envoltorios y mediador para realizar los procesos de optimización. Para más detalle sobre este tema, consúltese el apartado II.5.2.2.

Tal y como ya se ha comentado previamente, las técnicas utilizadas para las tareas 4) y 5) son fuertemente dependientes del tipo de fuente sobre la que opera el envoltorio.

Los tipos de fuentes que fundamentalmente han atraído la atención hasta el momento son:

- Bases de datos convencionales (relacionales, orientadas a objetos, etc.). El acceso (tarea 4) se realiza a través de alguna interfaz de establecimiento de conexiones y emisión de consultas. Esta interfaz puede ser específica del fabricante o bien seguir alguna norma multi-fabricante, como JDBC[JDB01]. Las respuestas de estas fuentes son devueltas a través de la conexión abierta con la base de datos y son estructuradas, por lo que el problema de obtener las tuplas respuesta a la consulta (tarea 5) son triviales.
- Fuentes web. El acceso se realiza a través de una o varias peticiones sobre la fuente utilizando el protocolo http. La salida proporcionada para la consulta será habitualmente un documento HTML. Las tuplas que constituyen la respuesta a la consulta efectuada no tienen, sin embargo, porque estar contenidas por entero en ese documento, sino que puede ser necesario realizar algún tipo de navegación a través de un grafo de documentos HTML conectados entre sí por hiperenlaces o formularios.
- Documentos XML. El acceso a los mismos puede realizarse a través del protocolo http al igual que en el caso de las fuentes web, o bien puede utilizarse para ello algún otro procedimiento dependiendo de la interfaz de acceso al servidor que contiene los documentos. El proceso de extracción de las tuplas que constituyen la respuesta a la consulta es mucho más sencillo que en el caso de las fuentes HTML, ya que, en este caso, las tuplas y sus atributos aparecen claramente identificados en el documento o conjunto de documentos.
- Documentos de texto semiestructurado. En esta categoría se incluyen todos aquellos documentos que siguen una cierta estructura, aunque sea irregular y se carezca de un esquema describiéndola. Un ejemplo puede ser el catálogo de productos en formato imprimible de una compañía. El acceso a los documentos en este caso dependerá fundamentalmente del formato en el que esté almacenado el

documento, que a menudo será propietario de un determinado fabricante. El problema de extracción de las tuplas es similar al de las fuentes web y, a menudo, podrán utilizarse las mismas técnicas para ambos casos.

En el caso de envoltorios sobre bases de datos convencionales, dado que la emisión de consultas sobre la fuente y la extracción de las tuplas de respuesta se realiza de acuerdo a interfaces de programación claramente definidos, el único problema a considerar es la traducción de la consulta recibida desde el lenguaje de consulta del mediador al lenguaje de consulta de la fuente.

Esta tarea consiste en la conversión de una expresión escrita en un determinado lenguaje estructurado (lenguaje del mediador) a una expresión equivalente en otro lenguaje estructurado diferente (lenguaje de la fuente). Por lo tanto, la complejidad del proceso para lenguajes de consulta basados en el mismo paradigma (e.g. modelo relacional) es habitualmente un proceso bastante simple (recuérdese que las posibles diferencias en las capacidades de consulta han sido ya resueltas al llegar a este punto).

Cuando el lenguaje de consulta de la fuente utiliza un paradigma diferente al del mediador (e.g. modelo relacional y modelo orientado a objetos), las dificultades pueden ser mayores, aunque sigue siendo más sencillo de automatizar que el caso en que los lenguajes de consulta no presentan una estructura clara. Este problema ha sido estudiado fundamentalmente en el ámbito de los sistemas de bases de datos federadas y existen ya algunos sistemas comerciales[IBM98] que lo abordan para un amplio número de casos.

En el caso de los documentos de texto semiestructurado, normalmente la fuente no proporciona por sí misma un lenguaje de consultas, sino que el mediador sólo puede extraer todas las tuplas y posteriormente aplicar él mismo la consulta efectuada. El método de acceso utilizado para acceder al documento dependerá del formato en el que esté almacenado el documento y puede ser propietario, aunque normalmente permitirá un acceso sencillo al mismo a través de alguna interfaz de programación. El proceso de extracción de las tuplas contenidas en el documento puede ser abordada con técnicas similares a las necesarias en fuentes web y su discusión se aplaza hasta el momento de tratar dicho tipo de fuentes.

Las fuentes basadas en documentos XML pueden tratarse como un subconjunto de las de documentos de texto semiestructurado donde el problema de extracción de tuplas es considerablemente más simple debido a que existe un esquema que describe como se disponen dichas tuplas en el documento, lo que convierte el problema en trivial.

Sin embargo, la aparición de lenguajes de consulta específicos para documentos XML, de los cuales quizás XQuery[CFRS01] sea el principal exponente, permite también tratar este caso como un subconjunto de las fuentes basadas en bases de datos convencionales. Sin embargo, en este caso, el problema de traducción de la consulta desde el lenguaje del mediador al de la fuente presenta diferencias significativas puesto

que los documentos XML pueden tener algunas características de los datos semiestructurados y los lenguajes de consulta sobre ellos presentan importantes variaciones con respecto a los habituales en bases de datos convencionales. Por lo tanto, si el mediador utiliza un lenguaje de consulta rígido, similar a los de bases de datos convencionales, el proceso de traducción será más complejo.

Sin duda, el tipo de fuentes que ha recibido más atención hasta la fecha en lo que se refiere a la generación de envoltorios para sistemas mediadores, son las fuentes web. Por la extensión con que han sido tratados y la importancia práctica que tienen en los sistemas mediadores modernos, este problema es analizado en profundidad en el siguiente subapartado.

II.5.4.1.1. ENVOLTORIOS PARA FUENTES WEB

En este apartado se analizan las principales dificultades específicas para la creación de envoltorios sobre fuentes web. Se exponen también las ideas básicas que subyacen en las principales técnicas que han sido propuestas hasta el momento para abordar estas dificultades.

En primer lugar se examinan los principales problemas encontrados en lo que se refiere a la tarea 4) de la lista de tareas de los envoltorios: la conversión de la consulta recibida del mediador en un acceso a una fuente que tenga como resultado la obtención de un documento o conjunto de documentos conteniendo las tuplas que constituyen la respuesta a la consulta efectuada.

Posteriormente, se aborda el problema de cómo extraer las tuplas deseadas de los documentos obtenidos en el paso anterior (tarea 5).

II.5.4.1.1.1. Traducción de consultas y acceso a fuentes web

Acceder a las páginas que contienen la respuesta a una consulta en una fuente web involucrará, en el caso más sencillo, rellenar automáticamente una serie de campos de un formulario HTML y enviar la correspondiente petición HTTP a la fuente. Este paso puede automatizarlo el programa envoltorio estableciendo una conexión HTTP con el servidor web y construyendo una petición con los parámetros adecuados.

Ejemplo II.5.4.1: Consulta a través de HTTP

Considérese una fuente web que permite la consulta de una base de datos de referencias bibliográficas, a la que llamaremos `R`. Para la consulta de dicha base de datos, la fuente ofrece un formulario HTML de consulta con dos campos, que permiten consultarla por los atributos `título` y `autor`. Supongamos que el envoltorio recibe del mediador la consulta (expresada en sql):

```
select * from R where (titulo='Corazón tan Blanco') and
(autor='JavierMarías')
```

La traducción de esta consulta al "lenguaje" de la fuente consiste en construir una petición http al servidor web de la fuente equivalente a la que construiría automáticamente un navegador de Internet cuando un usuario rellenase el campo `título` del formulario con el valor '`Corazón tan Blanco`', el campo `autor` con el valor '`Javier Marías`' y pulsase el botón `Enviar` del formulario.
□

Varios de los sistemas de ayuda a la generación de envoltorios para fuentes web proporcionan clientes http embebidos de diversa complejidad que permiten construir peticiones de este tipo[GRV98][HGC97][KM98] [LPH00].

En algunos casos [GRV98], se proporcionan también mecanismos simples para, partiendo de cierta metainformación especificada de manera declarativa por el creador del envoltorio, realizar de manera automática la transformación desde las consultas expresadas en el lenguaje del mediador a las peticiones HTTP equivalentes.

Ejemplo II.5.4.2: Transformación de una consulta del mediador en una petición HTTP

Considérese el mismo caso del ejemplo anterior. Supóngase que el servidor web de la fuente tiene la dirección `http://www.nombrefuente.com`, el cgi[HTT97] asociado al formulario se encuentra en la ruta relativa `/cgi-bin/` y su nombre es `nombrecgi.cgi`. Los parámetros esperados por el cgi son tres, el primero de ellos `campotitulo` debe rellenarse con el título que se desea consultar y el segundo `campoautor` debe rellenarse con el autor. El tercero es un parámetro con valor fijo que el servidor utiliza para propósitos particulares. Una manera en la que el envoltorio podría expresar la metainformación referente al formulario sería:

```
http://www.nombrefuente.com/nombrecgi.cgi?campotitulo=$TITULO&campoau
tor=$AUTOR&paramFijo=valorFijo
```

Como puede verse, la notación utilizada es muy similar a la que se usa habitualmente para representar peticiones http de tipo GET[HTT97]. La única diferencia es que el valor asignado a cada uno de los campos del formulario es un identificador (prefijados en este caso con el símbolo '`$`') que serán, en tiempo de ejecución, sustituidos por el valor para los atributos `título` y `autor` que aparezcan en la consulta recibida del mediador. Así, si la consulta es:

```
 select * from R where (titulo='Corazón tan Blanco') and
(autor='JavierMarías')
```

el envoltorio puede realizar dichas sustituciones dinámicamente obteniendo así:

```
http://www.nombrefuente.com/nombrecgi.cgi?campotitulo=Corazón+tan+Bla
nco&campoautor=Javier+Marías&paramFijo=valorFijo
```

Esta petición puede ser pasada ya directamente al cliente http para que obtenga la página de respuesta a la consulta.

□

De esta manera, tomando como entrada una sencilla especificación declarativa, el envoltorio puede automatizar el proceso de traducción de consultas y obtención de las páginas HTML de respuesta.Desgraciadamente, en muchas fuentes web, especialmente en las comerciales, este esquema tan simple es claramente insuficiente.

Un reciente informe industrial sobre Integración de Contenidos [SH01], destacaba esta importante carencia de los sistemas de generación de envoltorios para fuentes web presentados hasta al momento diciendo: "la extracción de información desde fuentes web comerciales no consiste solamente en técnicas inteligentes de análisis sintáctico, sino también en tratar con las dificultades de navegar por páginas que contienen Javascript o HTML dinámico y requieren cookies o contraseñas".

A continuación, se analizan con más detalle estos problemas.

Una primera dificultad surge porque no siempre el proceso de consulta se realiza en un solo paso. A menudo el resultado de rellenar un primer formulario de búsqueda es un segundo formulario que permite restringir la búsqueda deseada. Por ejemplo, una fuente conteniendo tuplas representando ofertas de empleo, puede presentar un primer formulario que permita escoger al usuario el tipo de empleo en el que está interesado y, en función de su elección, le presentará un segundo formulario en el que podrá restringir más su búsqueda de acuerdo a características específicas de la profesión escogida en el primer paso de la búsqueda.

Para tratar esto, se precisa que el envoltorio sea capaz de manejar secuencias de navegación compuestas de varias rutas como la especificada en el ejemplo anterior. No es difícil extender el esquema anterior para contemplar estos casos, si bien no todos los sistemas mencionados anteriormente lo soportan.

Otra dificultad surge de la práctica muy extendida en las fuentes web actuales, de construir rutas http de forma dinámica utilizando funciones Javascript [JAS]. Por ejemplo, los enlaces y formularios de muchas páginas web actuales, no tienen una URL en su campo `href` o `action`, sino una función Javascript que realiza una computación para obtener la URL final. Para poder construir una ruta http que permita al cliente http conectarse a la página a la que conduce un enlace de ese tipo, el creador del envoltorio debe realizar código *ad-hoc* para reproducir el funcionamiento de la función Javascript. También puede delegar esta tarea en un intérprete Javascript, pero entonces debe asegurarse de implementar los objetos Javascript que habitualmente un navegador proporciona, ya que el código Javascript incluido en las páginas web habitualmente asume que dichos objetos están disponibles. También debe asegurarse de descargar todos los marcos (o *frames*)[HTM99] de la página, incluso aquellos ocultos que no se traducen en ninguna representación gráfica para el usuario, ya que pueden contener funciones u objetos Javascript referenciados por las funciones que

crean las URLs. En conjunto este proceso, realizado manualmente, llega a ser extremadamente complejo incluso para programadores muy expertos.

Otro problema grave de este esquema se deriva de las técnicas de "mantenimiento de sesión" utilizadas por cada vez más sitios web.

Esto quiere decir que el programa envoltorio antes de invocar directamente la ejecución de un formulario HTTP puede precisar realizar algún paso previo para el establecimiento de una "sesión" en la fuente. Por ejemplo, en muchos sitios web una invocación del formulario de consulta sólo funcionará correctamente si el usuario ha pasado previamente por su página de entrada, momento en el que se inicia implícitamente una sesión mantenida o bien mediante "cookies"[HTT97], o bien mediante identificadores de sesión. Esto sirve a los sitios web para rastrear el comportamiento de los usuarios y ligar entre sí peticiones http que, de lo contrario, no podrían ser identificadas como realizadas por un mismo usuario.

El establecimiento y mantenimiento de sesiones basadas en 'cookies' está normalizado por el protocolo http y, por lo tanto, puede ser manejado de manera automática e idéntica para todas las fuentes, por lo que no constituye un problema serio (más allá del esfuerzo extra que es necesario realizar en la programación del cliente http). Sin embargo, el caso de los identificadores de sesión es más complicado y, desgraciadamente para nuestros propósitos, es ampliamente utilizado en los sitios web.

Un identificador de sesión es un número único que el servidor genera aleatoriamente cuando se crea una nueva sesión por parte de un usuario y que queda asociado unívocamente a dicha sesión hasta que esta expira (normalmente una sesión expira automáticamente tras transcurrir un cierto tiempo en el que el usuario ha estado inactivo). En el momento en que la sesión expira, su número asociado deja de ser válido.

El protocolo http no especifica una manera normalizada de generar y mantener a lo largo de la sesión este número, por lo que cada fuente que los utiliza realiza su propia implementación ad-hoc. El mecanismo básico consiste en que todas las páginas por las que navegue el usuario en la fuente tendrán una parte generada dinámicamente conteniendo el número de sesión asignado. Cada vez que el usuario navega a una nueva página a través de un hiperenlace o un formulario, el número de sesión se le pasa al nuevo documento a través de la invocación http mediante un parámetro de tipo oculto (`hidden`). De esta manera, toda la navegación realizada por el usuario en el sitio web, queda asociada al número de sesión.

Ejemplo II.5.4.3: Una fuente con identificadores de sesión

Siguiendo con los ejemplos anteriores, supóngase ahora que la fuente utiliza números de sesión. Esto quiere decir que existirá un cuarto parámetro (al que llamaremos *numsesion*) esperado por el cgi y cuyo valor será el número de sesión asignado. Así, el formato de una petición sobre el formulario será ahora:

```
http://www.nombrefuente.com/nombrecgi.cgi?campotitulo=$TITULO&campoau
tor=$AUTOR&paramFijo=valorFijo&numsesion=numero
```

donde `numero` es el número de sesión. El problema deriva de que `numero` no es un valor fijo sino que será diferente para cada sesión, por lo que debe ser obtenido por el envoltorio y rellenado en tiempo de ejecución. La metainformación almacenada por el envoltorio sobre el formulario podría entonces ser la siguiente:

```
http://www.nombrefuente.com/nombrecgi.cgi?campotitulo=$TITULO&campoau
tor=$AUTOR&paramFijo=valorFijo&numeroSesion=$NUMERO
```

con la importante diferencia de que el valor para el identificador `$NUMERO` no puede ser obtenido por el envoltorio de la consulta recibida del mediador sino que debe obtenerlo por sus propios medios.

□

El problema es que, al no ser parte de la norma http, un cliente de dicho protocolo, como los proporcionados por los sistemas mencionados anteriormente, no puede obtener de manera automática los números de sesión. Por lo tanto, la única manera por la que el envoltorio puede conseguirlo es extrayéndolo directamente mediante un proceso ad-hoc de la página HTML de entrada a la fuente (la página de entrada a la fuente es la única a la que el envoltorio puede conectarse sin haber establecido una sesión, por lo que es la única que puede usar para extraer el número de sesión). En dicha página de entrada el número de sesión estará insertado en algún parámetro oculto que se pasa al servidor cada vez que se invoca un enlace o formulario desde la página. Nuevamente, el lugar de la página en la que se encuentra o la sintaxis utilizada para especificarlo es totalmente particular de cada fuente. Por lo tanto, el envoltorio tiene que realizar el siguiente proceso:

1. El cliente http se conecta a la página de entrada de la fuente
2. Un programa construido (normalmente de manera totalmente ad-hoc) por el generador del envoltorio, extrae de la página HTML obtenida, el número de sesión asignado (que está presente en la página como un parámetro oculto). Este paso no puede automatizarse ya que, como se ha comentado, es algo que no está normalizado por el protocolo http y que cada fuente realiza de manera particular.
3. Ahora el número puede sustituirse por el identificador `$NUMERO` para obtener una petición http válida en la fuente.

Este esquema se complica aún más porque algunas fuentes utilizan esquemas de mantenimiento sesión más complejos. Por ejemplo, en algunas fuentes el número de sesión se obtiene a partir de varios parámetros ocultos diseminados por la página html que se fusionan en un único número de sesión mediante la computación de una función JavaScript. En ese caso, el programador del envoltorio debe realizar los siguientes pasos:

1. El cliente http se conecta a la página de entrada de la fuente.
2. Un programa construido (normalmente de manera totalmente ad-hoc,) extrae de la página HTML obtenida, todos los fragmentos del número de sesión asignado (que están presentes en la página como parámetros ocultos). Para complicar más el problema, algunas fuentes incluso distribuyen los fragmentos en marcos (*frames*) diferentes. Nuevamente, este complejo paso no puede automatizarse ya que es particular de cada fuente.
3. El programador construye un programa equivalente al código Javascript que se encuentra en la página, de manera que combine adecuadamente los fragmentos del número de sesión para obtener el número de sesión real.
4. A las sucesivas peticiones http que se realicen sobre la fuente en esa sesión, se añade siempre el número obtenido en 3).

Como fácilmente puede deducirse, en estas condiciones el proceso de creación y mantenimiento de envoltorios es largo, tedioso y requiere código ad-hoc difícil de generar y mantener incluso para programadores expertos.

Es importante notar que para un navegador de Internet, todo esto no supone realmente un problema ya que funciona de un modo interactivo: un navegador no necesita preocuparse por el hecho de que el formato utilizado para realizar una determinada consulta sobre un formulario cambie en cada ocasión en que la realiza, su única preocupación es resolver la consulta enviada *en ese momento* por el usuario. Es responsabilidad del servidor tratar los números de sesión a su conveniencia para devolver el resultado esperado por el usuario.

Sin embargo, un envoltorio tiene que preocuparse de disponer de un método de traducción de consultas uniforme que funcione en todos los casos con la misma metainformación.

Las complicaciones a abordar no terminan ahí. En otros sitios web puede ser necesario realizar algún proceso de autenticación, por ejemplo mediante el esquema usuario/contraseña. También es importante tener en cuenta que muchas fuentes utilizan el protocolo HTTPS para proporcionar el acceso a algunas de sus páginas.

Por lo tanto, la simple inclusión en un sistema mediador de funcionalidades basadas en clientes http simples, no son suficientes para tratar la mayor parte de las fuentes (al menos, de las comerciales), y es necesario disponer de mecanismos más sofisticados. Este es, a nuestro juicio, una de las principales limitaciones de los sistemas de generación de envoltorios propuestos hasta el momento, tal y como ya fue recalcado en [SH01].

El sistema presentado en esta tesis doctoral incorpora un novedoso mecanismo que, mediante el uso transparente del popular navegador de Internet Microsoft Internet Explorer[IEX01] permite independizar totalmente al envoltorio de las complejidades del manejo de sesiones, tratamiento de Javascript, etc. convirtiendo este paso en puramente mecánico y, por lo tanto, automatizable para todas las fuentes.

II.5.4.1.1.2. Extracción de tuplas desde fuentes web

En cuanto a la tarea de extracción de las tuplas resultado desde las páginas HTML devueltas, la mayor dificultad estriba en que las páginas HTML han sido diseñadas para ser vistas por humanos, y no para facilitar su procesamiento automático por parte de programas.

Debido a esto, los datos normalmente están embebidos en texto en lenguaje natural u ocultos bajo representaciones gráficas. La estructura de HTML poco aporta en este sentido ya que proporciona información únicamente sobre la forma de presentar los datos en un navegador de Internet, pero no dice nada sobre su contenido o estructura.

Además, la forma de las páginas web cambia frecuentemente (esto es especialmente cierto en los sitios web explotados comercialmente), haciendo complicado el mantenimiento de los programas envoltorio.

Las herramientas creadas hasta el momento para facilitar la creación de este tipo de programas pueden clasificarse en tres categorías fundamentales: lenguajes de definición de gramáticas especializadas, técnicas de aprendizaje inductivo, y herramientas gráficas para la creación "supervisada" de envoltorios. En los siguientes subapartados se trata, respectivamente, cada uno de estos enfoques y, finalmente, son comparados entre sí.

II.5.4.1.1.2.1. Técnicas basadas en lenguajes de definición

Esta clase de herramientas se basa en proporcionar algún tipo de lenguaje de especificación que pretende que la generación de gramáticas especializadas para extraer los datos deseados desde la página, sea más fácil y rápida.

La idea básica es que el lenguaje de especificación, por un lado, proporcione ya construidas funcionalidades comunes para estos casos, tales como analizadores sintácticos de HTML o funciones para moverse a través de los elementos de la página.

Algunos sistemas más avanzados utilizan además heurísticas inherentes a la forma de presentar los datos en HTML, lo que permite simplificar los lenguajes de definición, haciéndolos más asequibles a personal de menor cualificación.

[HGC97] fue una de las primeras propuestas de este tipo de sistemas, creada dentro del marco del proyecto TSIMMIS.

El sistema de extracción de tuplas de TSIMMIS analiza las páginas HTML basándose en un fichero de configuración, que es específico para cada fuente.

Este fichero consiste en una secuencia de comandos, cada uno definiendo un paso de extracción. Cada comando es de la forma: `[variable, fuente, patrón],`

donde la fuente especifica el texto de entrada a ser considerado, el patrón indica como encontrar el texto de interés dentro de la fuente, y las variables son una, o más, variables de extracción que contendrán los resultados. El texto encajado en las variables se puede usar como entrada para los siguientes comandos (así, si por ejemplo, una variable contiene una URL, se puede utilizar esa URL para navegar a ella y seguir extrayendo tuplas). Después de que se ejecute el último comando, algún subconjunto de las variables contendrá los datos de interés. El contenido de esas variables se empaqueta en objetos que pueden ya ser utilizados por el mediador de TSIMMIS.

Ejemplo II.5.4.4: El sistema de extracción de tuplas de TSIMMIS

Veamos un ejemplo de cómo sería un fichero de especificación para un sitio web llamado Intellicast, que contiene información meteorológica de las principales ciudades europeas en una tabla con el siguiente aspecto.

		Tue, Jan 28, 1997		*Wed, Jan 29, 1997*	
country	*city*	*forecast*	*hi/lo*	*forecast*	*hi/lo*
Austria	Viena	snow	-2/-7	snow	-2/-7
Belgium	Brussels	ptcldy	3/-4	ptcldy	3/-4
Czech Republic	Prague	snow	-1/-7	snow	-1/-7
Denmark	Copenhagen	fog	3/-1	fog	3/-1
England	Birmingham	ptcldy	9/-3	ptcldy	7/3
England	Liverpool	ptcldy	8/2	ptcldy	6/2
England	London	ptcldy	9/0	ptcldy	8/4
England	Manchester	ptcldy	8/-1	ptcldy	6/3
England	Plymouth	ptcldy	9/3	ptcldy	8/5

y cuyo código HTML es:

```
 1 <HTML>
 2 <HEAD>
 3 <TITLE>INTELLICAST: europe weather</TITLE>
 4 <A NAME="europe"></A>
 5 <TABLE BORDER=0 CELLPADDING=0 CELLSPACING=0 WIDTH=509>
 6 <TR>
 7 <TD colspan=11><I>Click on a city for local forecasts</I><BR></TD>
 8 </TR>
 9 <TR>
10 <TD colspan=11><I> temperatures listed in degrees celsius
</I><BR></TD>
11 </TR>
12 <TR>
13 <TD colspan=11><HR NOSHADE SIZE=6 WIDTH=509></TD>
14 </TR>
15 </TABLE>
16 <TABLE CELLSPACING=0 CELLPADDING=0 WIDTH=514>
17   <TR ALIGN=left>
18     <TH COLSPAN=2><BR></TH>
19     <TH COLSPAN=2><I>Tue, Jan 28, 1997</I></TH>
```

```
20      <TH COLSPAN=2><I>Wed, Jan 29, 1997</I></TH></TR>
21    </TR>
22    <TR ALIGN=left>
23      <TH><I>country</I></TH>
24      <TH><I>city</I></TH>
25      <TH><I>forecast</I></TH>
26      <TH><I>hi/lo</I></TH>
27      <TH><I>forecast</I></TH>
28      <TH><I>hi/lo</I></TH>
29    </TR>
30    <TR ALIGN=left>
31      <TD>Austria</TD>
31    <TD><A HREF=http://www.intellicast.com/weather/vie/>Vienna</A>
32    </TD>
33      <TD>snow</TD>
34      <TD>-2/-7</TD>
35      <TD>snow</TD>
36      <TD>-2/-7</TD>
37    </TR>
38    <TR ALIGN=left>
39      <TD>Belgium</TD>
40      <TD><A
HREF=http://www.intellicast.com/weather/vie/>Brusels</A></TD>
41      <TD>fog</TD>
42      <TD>2/-2</TD>
43      <TD>sleet</TD>
44      <TD>3/-1</TD>
45    </TR>
        .
        .
    </TABLE>
      .
  </HTML>
```

El fichero de especificación que se utiliza es el siguiente:

```
 1 [["root",
 2   "get ('http://www.intellicast.com/weather/europe/')",
 3   "#"
 4  ],
 5  ["temperatures",
 6   "root",
 7   "*<TABLE*<TABLE*</TR>#</TABLE>*"
 8  ],
 9  ["_citytemp",
10   "split (temperatures,'<TR ALIGN=left>')",
11   "#"
12  ]
13  ["city_temp"
14   "_citytemp[1:0]",
15   "#"
16  ]
17 ["country,c_url,city,weath_tody,hgh_tody,low_today,weath_tomorrow,
      hgh_tomorrow,low_tomorrow",
18   "city_temp",
```

```
19
"*<TD>#</TD>*HREF=#>#</A>*<TD>#</TD>*<TD>#/#</TD>*<TD>#</TD>*<TD>#/#*
"
20  ]]
```

La lista de comandos está encerrada entre los símbolos '[' y ']', y cada comando, a la vez, está delimitado por esos mismos símbolos. El proceso de extracción del ejemplo se realiza por medio de cinco comandos. El comando inicial (líneas 1-4) obtiene los contenidos del fichero fuente cuya URL se especifica en la línea 2, en la variable `root`. El carácter '#' de la línea 3 significa que se va a considerar el contenido completo del fichero. Una vez que se ha obtenido dicho contenido y se ha leído en `root`, el extractor filtrará los datos no deseados tales como las marcas HTML y texto extra, con los otros cuatro comandos.

El segundo comando (líneas 5-8) especifica que el resultado de aplicar el patrón de la línea 7 a la variable de fuente `root`, se almacena en una nueva variable llamada `temperature`. El patrón se puede interpretar como sigue: "descarta todo hasta la primera ocurrencia del token `</TR>` ('*' significa descartar) en la segunda definición de una tabla, y guarda los datos que están entre `</TR>` y `</TABLE>` ('#' significa guardar)". Los dos tokens `<TABLE` entre el '*' se utilizan como ayuda de navegación para identificar el token `</TR>` correcto, ya que no hay manera de especificar un número de ocurrencia de un token (como por ejemplo el tercer `</TR>`). Después de este paso, la variable `temperatura` contiene la información que está en las líneas 22 y siguientes en el fichero con el código HTML, hasta el token `</TABLE>`.

El tercer comando (líneas 9-12) le indica al extractor que divida el contenido de la variable `temperatura` en fragmentos de texto, usando la cadena `<TR ALIGN=left>` (líneas 22, 30, 38, etc. en el código HTML) como delimitador de los fragmentos. Cada fragmento representa una fila de la tabla de temperaturas. El resultado de la división se almacena en una variable temporal llamada `_citytemp` (el subrayado del principio del nombre indica que es una variable temporal, y que su contenido no se incluye en el objeto OEM resultado). El operador `split` sólo se puede utilizar si la entrada está constituida por fragmentos estructurados de la misma forma, con un delimitador separando esos fragmentos. Si se piensa en las variables como listas (las vistas hasta ahora tendrían un solo elemento), entonces el resultado del operador de división puede verse como una nueva lista, con tantos miembros como filas hay en la tabla de temperaturas. Cuando se aplica un patrón a una variable, significa que se aplica el patrón a cada miembro de la variable.

En el comando 4 (líneas 13-16), el extractor copia el contenido de cada elemento del vector temporal en el vector `city_temp` comenzando en el segundo elemento desde el principio. El primer entero de la instrucción `_citytemp[1:0]` indica el comienzo de la copia (ya que los índices empiezan en 0, 1 se refiere al segundo elemento), y el segundo entero indica el último elemento incluido (contando desde el final). Como resultado de esta operación, se ha excluido la primera fila de la tabla, que contiene las cabeceras. También se podía haber eliminado esta primera fila en el

segundo comando, especificando una condición `*</TR>` adicional antes del '#' de la línea 7. El comando final (líneas 17-20) extrae los valores individuales para cada elemento del array `city_temp` y se los asigna a las variables listadas en la línea 17 (`country`, `c_url`, `city`, etc.).

La salida del proceso sería un objeto `temperature`, compuesto por una lista de objetos `city_temp`. Cada objeto `city_temp` contiene todos los datos para una ciudad.

Entre las características positivas de este sistema, hay que mencionar que construir envoltorios con él es mucho más rápido que hacerlo con los lenguajes de programación habituales. Además, a diferencia de los sistemas de aprendizaje inductivo, es adecuado aunque las fuentes sean muy complejas: la especificación se complicará pero las fuentes podrán ser tratadas sin mayores complicaciones.

Entre las características negativas, nos encontramos con que los ficheros de especificación son bastante complejos. Claramente parece necesario que sean construidos por programadores.

Además las especificaciones son muy dependientes del código HTML de las páginas, y si cambia prácticamente cualquier parte de ellas, aunque el cambio no afecte directamente a los datos que queremos extraer, será necesario modificar la especificación. Esto es especialmente grave en el caso de fuentes comerciales que cambian su apariencia muy frecuentemente. Por ejemplo, bastaría que en el código html anterior se insertase una nueva tabla al comienzo (por ejemplo para incluir un anuncio publicitario) para que el envoltorio dejase de funcionar, cuando en realidad la estructura de los datos a extraer no habría cambiado.

Otro problema, aunque relacionado más con la tarea de acceso, es que el mecanismo utilizado para acceder a las páginas (un simple cliente http accesible a través del comando GET) presenta todos los problemas derivados de la necesidad de mantenimiento de sesión, tratamiento de Javascript, etc. comentados en el apartado II.5.4.1.1.1

□

Otros sistemas que utilizan el enfoque de lenguajes de especificación son [GRV98][KM98].

En [GRV98], para el problema particular de la extracción de tuplas, se dispone de un lenguaje llamado QEL (*Qualified path expresión Extractor Language*) que permite definir extractores simples. Otro lenguaje, llamado CESL (*Complex Extractor Specification Language*) permite combinar diversos extractores simples para construir objetos complejos.

QEL permite definir consultas declarativas sobre el árbol HTML de un documento. Una consulta con QEL permite:

- Identificar los elementos a extraer dentro de un documento HTML basándose en sus contenidos o en su lugar en el documento.
- Extraer datos de los elementos seleccionados.

El sistema proporciona también operadores ya construidos para manipular los datos extraídos. Además, proporciona un operador llamado `construct` que permite construir objetos estructurados basándose en los datos extraídos, facilitando así la interacción con los mediadores.

Para ilustrar mejor el funcionamiento del sistema, considérese el siguiente ejemplo:

Ejemplo II.5.4.5: Extracción de tuplas desde fuentes web con [GRV98]

Supóngase un documento HTML conteniendo una tabla cuyo título es *'Galicia'*. Cada fila de la tabla se corresponde con una de las provincias de dicha comunidad autónoma. La segunda columna de cada fila contiene el nombre de la ciudad. Supóngase que deseamos utilizar la fuente para extraer el nombre de todas las provincias gallegas:

```
ProvinciasGalicia: Root.child[Name=table &
Title="Galicia"].child.child[Name=td & Occurrence=2].Data
```

Esta consulta comienza identificando el objeto TABLE correcto utilizando los parámetros `Name` y `Title`: concretamente busca las tablas del documento cuyo título sea "`Galicia`". Al aplicar una vez el operador `child` se obtienen todas las filas de la tabla. Por lo tanto `Root.child[Name=table & Title="Galicia"].child` es una lista de objetos, cada uno de los cuales es una fila de la tabla.

Por lo tanto, aplicando nuevamente el operador `child` obtenemos todas las columnas de cada una de las filas. Con el parámetro `Ocurrence=2`, se indica que las columnas deseadas son las situadas en segundo lugar. Con `.Data` se accede al dato contenido en esas columnas.

Así, finalmente, se obtienen los datos deseados. Ahora `ProvinciasGalicia` podría ser utilizado dentro de una operación con el operador `construct`, para crear un objeto conteniendo los datos extraídos.

□

Por su parte, CESL es necesario para construir extractores más complejos que incluyan funcionalidades tales como la extracción de tuplas desde múltiples documentos o la creación de objetos complejos que obtengan sus datos de múltiples extractores.

La evaluación que puede realizarse de este sistema es similar a la ya expuesta para el sistema de extracción de TSIMMIS. En el lado positivo, construir envoltorios con el sistema es mucho más rápido que hacerlo con los lenguajes de programación habituales y, a diferencia de los sistemas de aprendizaje inductivo, es adecuado aunque las fuentes sean complejas.

Sin embargo, nuevamente nos encontramos con que los ficheros de especificación son bastante complejos y deben ser construidos y mantenidos por programadores.

Una ventaja con respecto al sistema de TSIMMIS es que, al permitir funcionalidades como la búsqueda de elementos por nombre, las especificaciones no son tan dependientes del código HTML de las páginas, y en general se verán afectadas sólo por los cambios que involucren modificaciones en la disposición en la página de los datos extraídos en sí.

Para acceder a las páginas, se utiliza también el simple esquema basado en un cliente http, por lo que se mantienen los problemas comentados en el apartado II.5.4.1.1.1.

□

En [KM98] se presenta WebL, un lenguaje de programación de *scripts* que incorpora un importante número de construcciones para manejar la conexión a fuentes web y la extracción de tuplas estructuradas desde las mismas. WebL es actualmente propiedad de la empresa COMPAQ y recibe también el nombre de *COMPAQ Web Programming Language.*

WebL proporciona una serie de componentes JAVA, accesibles a través de un lenguaje de scripting, que implementan funcionalidades comunes para este tipo de tareas.

En el apartado de conexión a sistemas web, WebL proporciona un cliente http embebido y funcionalidades (llamadas *combinadores de servicios)* incluidas en el lenguaje para:

- Recargar automáticamente una página si la transferencia se para.
- Reintentar peticiones que hayan fallado.
- Terminar peticiones después de un tiempo máximo de espera.
- Lanzar peticiones en paralelo
- Si varias fuentes dan la misma información, el sistema permite lanzarles peticiones en paralelo, quedándose con la respuesta de la que conteste antes.

Estas funcionalidades pretenden replicar las estrategias utilizadas por los navegantes expertos cuando se conectan a una página determinada.

El problema de la complejidad de realizar los procedimientos de establecimiento de sesión y tratamiento de Javascript, tampoco es solucionado por este sistema, por lo que se requieren los complejos procesos combinados de conexiones HTTP y extracción de números de sesión que ya se han comentado.

En cuanto al proceso de extracción, WebL proporciona un álgebra que permite delimitar porciones del texto de una página mediante funciones específicas capaces de tratar con marcas HTML y texto visible. Así, una porción puede ser definida por la posición de dos marcas HTML o por una expresión regular expresada en notación Perl[Per99].

La posición de cada una de estas porciones está delimitada de manera que es posible determinar la posición relativa dentro de la página de una porción con respecto a otra (una porción puede estar antes, después, incluida en otra o solapada en otra).

Estas porciones pueden agruparse en conjuntos sobre los que pueden realizarse diversos tipos de operaciones:

- Operadores Básicos: uniones, intersecciones, exclusiones.
- Operadores Posicionales: permiten determinar si una porción o conjunto de porciones están antes, después o se solapan.
- Operadores Jerárquicos: permiten determinar si una porción o conjunto de porciones está contenida en otra.

Para dar una idea del aspecto de una especificación escrita con WebL, se incluye el siguiente ejemplo.

Ejemplo II.5.4.5: Extracción de tuplas con WebL

En este ejemplo se define una función que recibe como parámetro el símbolo de una cotización bursátil y se conecta a la web `fast.quote.com` para realizar una búsqueda por dicho símbolo y obtener su cotización. La cotización a extraer se encuentra situada dentro de una tabla cuya cabecera es la cadena "`Stock Quotes`", en la columna "`Last`". El aspecto de la página es similar al siguiente:

Stock Quotes							
Symbol	Time	Open	High	Low	Last	Change	Volume
YHOO	16:03	48 9/16	49 1/4	48 5/16	**49**	**+7/16**	933,600

El código WebL necesario para realizar la tarea se muestra a continuación:

```
1) Cotizacion:=fun (simbolo)
2) pagina:=getpage("http://fast.quote.com/fq/quotecom/quote",[.
symbols=simbolo .]);
3) (pagina.Elem ("B") in  (pagina.Elem ("TABLE") contain pagina. Pat
("Stock Quotes"))[0])[1].Text ()
4) end;

5) s := Cotizacion ("YHOO")
```

La función recibe el nombre `Cotizacion` y recibe el parámetro de nombre `simbolo`. Es definida en la línea 1) y se cierra en la línea 4. La línea 5 se limita a invocarla con el símbolo `"YHOO"`.

La línea 2 es la encargada de realizar la búsqueda en la fuente. Para ello usa la función `getpage` que es capaz de realizar una petición HTTP de tipo GET. El parámetro `symbols` del formulario de búsqueda es rellenado con el símbolo recibido como parámetro. La página obtenida se almacena en la variable `pagina`.

La línea 3 es la encargada de extraer la cotización. Lo que hace realmente es extraer el primer texto en negrita contenido dentro de la primera tabla de la página que contenga el texto `"Stock Quotes"`.

Con `pagina.Elem("TABLE")` se seleccionan todas las porciones delimitadas por las marcas `<TABLE>` y `</TABLE>` de la página, es decir todas las tablas. Con `pagina.Pat("Stock Quotes")` se seleccionan todas aquellas porciones de texto visible en la página que concuerdan con una expresión regular, en este caso la muy simple `Stock Quotes`. El operador `contain` entre ambas operaciones hace que el sistema se quede solamente con aquellas tablas que contengan en su interior alguna de las ocurrencias de `Stock Quotes`. Al aplicarle a ese nuevo conjunto el selector `[0]`, se selecciona sólo la primera de esas tablas.

Con `pagina.Elem("B")` se seleccionan todas las porciones delimitadas por las marcas `<B>` y `</B>`, es decir aquellos textos que estén en negrita. Al aplicar el operador *in* con este conjunto y la tabla obtenida previamente, se obtienen todas los textos en negrita contenidos en dicha tabla. Aplicando el selector `[1]` se selecciona sólo el primer texto en negrita. Aplicando la función `.Text()` se obtiene el texto de la cotización.

□

Como valoración, cabe decir que al igual que los sistemas anteriores, WebL sirve para tratar incluso fuentes muy complejas, pero a costa de requerir personal fuertemente cualificado para la creación de envoltorios. El lenguaje proporcionado es muy potente pero de muy bajo nivel.

Incluso para un caso trivial como el del ejemplo anterior, se requiere un código relativamente complejo. Con fuentes sólo algo más complejas, como tiendas web, las especificaciones precisas están sólo al alcance de programadores expertos. Además, como ya se ha dicho, un cliente http no es suficiente para manejar de forma sencilla los procedimientos de establecimiento de sesión de fuentes complejas.

Otro inconveniente es que WebL debe trabajar de manera síncrona, es decir que debe esperar a que la página se descargue totalmente antes de empezar el proceso de extracción. Esto impide que los motores de ejecución de consultas puedan ser asíncronos a su vez, es decir que puedan ir devolviendo resultados a medida que estos estén disponibles sin necesidad de esperar a que las fuentes hayan terminado.

Como ventaja con respecto a los sistemas previos, presenta sus construcciones para tratar situaciones habituales en fuentes web, tales como fallos de conexión, reintentos, etc.

□

Otros sistemas que siguen este enfoque y que también tienen interés son [SA99] y [All97].

II.5.4.1.1.2.2. Técnicas basadas en aprendizaje inductivo

Las técnicas de aprendizaje inductivo se basan en proporcionar a la herramienta un conjunto de páginas HTML etiquetadas que actúan como ejemplos. Por ejemplo, para una tienda virtual de libros, se tomarían varias páginas de respuesta de búsquedas y se etiquetarían marcando en la página de alguna manera cuáles son las tuplas a extraer y cómo se descomponen en sus atributos respectivos (por ejemplo, en tuplas que representasen referencias bibliográficas, indicando qué parte es un título de un libro, qué parte es un autor, etc.).

Partiendo de las páginas de ejemplo, la herramienta debe ser capaz de inducir la gramática subyacente que le permita extraer los datos de nuevas páginas. Por supuesto, el sistema funcionará mejor cuantos más ejemplos se le proporcionen. El reto está en conseguir sistemas que requieran el menor número posible de ejemplos y qué requieran la menor especialización que sea posible para realizar la elección de dichos ejemplos (a menudo un experto puede acelerar mucho el proceso escogiendo ejemplos relevantes).

Un buen estudio comparativo con alto nivel de detalle sobre los sistemas de este tipo puede encontrarse en [Eik99].

En este apartado se detallan solamente las características más relevantes de los principales sistemas.

En [DEW97] se muestra uno de los primeros sistemas basados en la idea de aprendizaje inductivo, llamado Shopbot.

Shopbot es, en realidad, una aplicación de propósito específico, destinada a extraer información solamente de tiendas electrónicas, y no de otros tipos de sitios web. Utiliza una mezcla de heurísticas específicas del dominio de la compra de productos y técnicas de aprendizaje inductivo para proporcionar un sistema que construye envoltorios de manera totalmente automática, sin intervención humana.

ShopBot opera en dos fases: la *fase de aprendizaje* y la fase de *compra comparativa*. En la fase de aprendizaje, la tienda electrónica es analizada para que el sistema sea capaz de "aprender" la descripción de la fuente. En la fase de compra comparativa, las descripciones aprendidas en la fase previa se utilizan para extraer información de la fuente.

Durante la fase de aprendizaje, se utilizan heurísticas simples para aprender automáticamente qué formulario se debe invocar y la manera de hacerlo. Para ello, el sistema se limita a navegar aleatoriamente por la fuente y va examinando las páginas que encuentra buscando formularios en su código HTML. Para cada formulario encontrado, intenta realizar consultas rellenando sus campos de manera aleatoria con una serie de consultas de ejemplo de las que dispone. Cuando en la respuesta a alguna de las consultas en algún formulario obtiene una página que parece contener varios resultados plausibles (el sistema también tiene una lista de respuestas esperadas a la consulta de ejemplo), asume que ha encontrado una manera correcta de consultar la fuente.

Nótese que esta técnica, además de ineficiente (aunque esto no es importante en la práctica ya que la fase de aprendizaje se realiza durante la creación del envoltorio y no durante la ejecución de consultas), utiliza la idea de que es posible proporcionar al sistema una lista de consultas de ejemplo que en todas las fuentes van a dar resultados muy parecidos. Esto es así hasta cierto grado en el caso concreto de tiendas electrónicas (por ejemplo, una búsqueda por la palabra `crichton` ofrecerá como resultado más o menos los mismos libros en todas las tiendas), pero obviamente no puede extenderse a otros dominios de aplicación. Además este sistema no funciona con tiendas que utilicen mecanismos de establecimiento de sesión, ni puede identificar correctamente todas las capacidades de consulta de las tiendas en casos que no sean muy simples (por ejemplo, no es capaz de tratar correctamente muchos formularios que incluyan listas de selección).

Una vez obtenida la manera de consultar las páginas, es necesario que el sistema aprenda la manera de extraer los resultados desde las páginas HTML.

Se asume que las páginas de resultado consisten de una cabecera, un cuerpo y una cola, dónde la cabeza y la cola son consistentes entre las diferentes búsquedas y el cuerpo contiene toda la información deseada. El formato de la página de resultados se determina en tres pasos.

En primer lugar se aprenden las respuestas proporcionadas por la fuente en situaciones en la que se produzcan consultas sin resultados o incorrectas. Esto puede hacerse mediante heurísticas como realizar consultas por valores imposibles (e.g. 'fjkdashf').

En segundo lugar, se realizan diversas consultas para detectar la cabeza y la cola. Se asume que éstas serán aquellas partes que se repitan al principio y al final de todas las páginas obtenidas.

Finalmente, se determina el formato del cuerpo de la página (que contiene la información buscada). Inicialmente, se generan una serie de posibles formatos abstractos de las descripciones de los productos. Cada uno de estos formatos abstractos es definido mediante una secuencia de marcas HTML y de cadenas de caracteres. Entonces, el sistema divide el cuerpo de la página en líneas lógicas y el algoritmo de

aprendizaje compara los diferentes formatos abstractos con las líneas lógicas para encontrar el que mejor concuerde, que será el utilizado finalmente. Para extraer el precio de los productos se utilizan heurísticas específicas.

Con todas estas técnicas, Shopbot consigue automatizar totalmente la generación de estos envoltorios para algunas tiendas electrónicas sencillas.

Sin embargo, esto tiene algo de engañoso en la práctica, ya que muchas tiendas no pueden ser tratadas por el sistema. Por poner un par de ejemplos, aquellas que usen sistemas de mantenimiento de sesión o que no se ajusten exactamente al esquema cabeza-cuerpo-cola, no podrán ser tratadas. Además, ya se ha comentado, que este sistema es sólo útil para la aplicación concreta de compra comparativa.

Además, incluso para funcionar con fuentes sencillas, el sistema requiere que cada vez que se añada una nueva categoría de productos, un experto proporcione una cuidadosamente escogida selección de ejemplos y de heurísticas relativas a la tiendas del producto a tratar. Los autores del sistema reconocen que construir bien esa selección de ejemplos es un proceso muy complejo.

□

WIEN[KWD97] (Wrapper Induction ENvironment) es una herramienta para la creación de envoltorios que parte de muchas de las ideas introducidas por Shopbot, para crear un sistema de aprendizaje inductivo para envoltorios, no restringido al dominio de la compra comparativa.

Para ello, en primer lugar, reduce su alcance, ya que abandona el problema de detectar y aprender formularios de consulta, centrándose sólo en la extracción de tuplas.

WIEN puede manejar lo que sus creadores denominan páginas HLRT. Una página HLRT es aquella que tiene una cabeza (Head), un conjunto de delimitadores a la izquierda (Left) y derecha (Right) de cada tupla a extraer y una cola (Tail).

El sistema trata de determinar estos elementos en cada página, lo que le permite primero obtener la porción de la página donde se encuentran las tuplas relevantes (suprimiendo la cabeza y la cola) y, posteriormente, delimitar cada una de las tuplas a extraer gracias a los delimitadores de izquierda y derecha.

Una vez realizado esto, es cuando trata de inducir los separadores que delimitan cada uno de los campos de las tuplas, obteniendo así finalmente los objetos a exportar.

Para automatizar el proceso lo más posible y no requerir que se etiqueten ejemplos para cada fuente nueva, WIEN (al igual que hacía Shopbot) toma como entrada una serie de heurísticas específicas del dominio de aplicación para generar automáticamente ejemplos etiquetados para las fuentes. La manera en que se obtengan esas heurísticas queda fuera del alcance de WIEN y es responsabilidad de su usuario. Si bien estas heurísticas tienen que ser identificadas y especificadas a WIEN por

personal experto, se tiene la ventaja de que si las heurísticas son adecuadas, no se precisa nueva intervención humana para ese dominio de aplicación.

El algoritmo de inducción toma como entrada los ejemplos etiquetados de una fuente y busca en el espacio de envoltorios definidos por el modelo HLRT. Para cada posible envoltorio HLRT, itera sobre todos los posibles delimitadores de campos dentro de una tupla. Cuando encuentra un envoltorio HLRT y un conjunto de delimitadores que satisfacen todos los ejemplos proporcionados, detiene la búsqueda. Para calcular el número de ejemplos que el algoritmo de aprendizaje debe examinar para asegurar que la probabilidad de fallo esté por debajo de un cierto límite, se utiliza un modelo basado en la teoría de aprendizaje computacional.

WIEN tiene algunas limitaciones importantes en cuanto al tipo de fuentes que puede tratar. Las fuentes que presenten alguna de las características siguientes no pueden ser tratadas por WIEN:

- Cómo ya se ha comentado, sólo puede tratar páginas del tipo HLRT
- Si las tuplas contienen atributos que a veces aparecen y a veces no.
- Si hay variaciones en el orden de los atributos.
- Si hay atributos con valores múltiples.

En concreto, de una lista de 30 fuentes web escogidas por los autores, WIEN fue capaz de tratar el 70%. La actual evolución en complejidad de las fuentes web (especialmente las comerciales) probablemente haría bajar ese porcentaje en la actualidad, ya que las características anteriores se encuentran en un número de fuentes cada vez mayor.

□

SoftMealy[HD98] surge para tratar de mejorar algunas de las limitaciones de WIEN y afirma tratar las fuentes que presentan tuplas con atributos opcionales, múltiples y con variaciones en el orden.

SoftMealy, a diferencia de WIEN, recibe como entrada un conjunto de ejemplos etiquetados de la fuente para la que se quiere construir el envoltorio, y utiliza un algoritmo de generalización inductiva para obtener reglas de extracción. El proceso de inducción analiza los ejemplos para determinar las posiciones de los elementos a extraer así como los delimitadores de los mismos, y las generaliza en el contexto proporcionado por los ejemplos, generando un conjunto de reglas generales de extracción.

El envoltorio inducido es un autómata no determinista donde los estados representan los hechos a ser extraídos y las transiciones entre estados representan reglas que definen los separadores entre ellos. Durante el proceso de extracción, el envoltorio va desencadenando transiciones en el autómata a medida que va reconociendo los separadores entre elementos.

Hay pocos datos sobre el comportamiento de SoftMealy con fuentes reales. Sus autores afirman que es capaz de tratar la mayor parte de las fuentes que los autores de WIEN identificaban como no tratables por sus sistema, si bien existen pocos datos publicados al respecto.

En particular, SoftMealy, aligual que WIEN, sigue asumiendo que los items de los que se desea extraer información pertenecen a una única clase de información con estructura plana. Muchas páginas en fuentes reales no siguen en la práctica esquemas tan simples. En muchos casos, la información a extraer está dispuesta jerárquicamente , lo cuál quiere decir que las tuplas a extraer contienen diferentes niveles de anidación (es decir que, dentro de una tupla, se encuentra una lista de *subtuplas* pertenecientes a una relación anidada dentro de cada tupla de la relación de nivel superior).

□

Dentro del sistema mediador Ariadne [KMAA98], se han desarrollado diversas soluciones para la generación semi-automática de envoltorios para páginas web.

El primer de ellos fue [AK97]. Este sistema de generación semi-automática de programas envoltorio utiliza heurísticas específicas de la presentación de documentos en HTML para inferir estructuras probables dentro de una página. Por ejemplo, al tratar un listado de características en una página HTML Ariadne puede suponer que un epígrafe escrito en una fuente de 14 puntos es de un nivel jerárquico superior a otro escrito en una fuente de sólo 12 puntos.

Sin embargo, conscientes de que estas heurísticas fallan con frecuencia, complementan el sistema con la intervención humana para confirmar o "desmentir" las suposiciones que el sistema ha hecho basándose en sus heurísticas. Se establece así un proceso iterativo entre el sistema y el operador del mismo en el que el primero, basándose en sus heurísticas y en la correcciones del segundo, va probando diversas gramáticas para extraer tuplas de la páginas, hasta que una de ellas es aceptada finalmente por el usuario como correcta, al ofrecer los resultados esperados.

Como resultado final del proceso, el sistema termina generando una gramática YACC que será la utilizada para extraer datos de las páginas con esa estructura.

El mismo equipo creó un nuevo sistema de generación de envoltorios llamado Stalker[KLMM99], y que ofrece características más avanzadas.

Stalker es otro método basado en aprendizaje inductivo que parte de una serie de ejemplos etiquetados de cada fuente, si bien las últimas versiones permiten cierto grado de entrenamiento automatizado. Al igual que SoftMealy, es capaz de tratar fuentes con atributos opcionales, múltiples y de orden variante. Además, sus extensiones más recientes le permiten tratar tuplas con niveles anidados. Entre sus limitaciones se encuentra el que no es capaz de extraer información insertada en las marcas html (como, por ejemplo, las url contenidas dentro de las marcas de tipo enlace).

Otra funcionalidad introducida recientemente en la generación de envoltorios para Ariadne es la auto-corrección de envoltorios. La idea básica es que, cuando se produce un cambio en una fuente que hace fallar el proceso de extracción, el sistema, ayudado por los ejemplos etiquetados previos y por técnicas cercanas a la disciplina de Recuperación de Información, intenta inferir una nueva etiquetación para las páginas con el nuevo formato. Si el proceso es exitoso, el sistema puede "autoentrenarse" automáticamente sin necesidad de que haya intervención humana.

Si bien, esta capacidad funciona por el momento sólo para ciertos casos simples, constituye el primer paso hacia la automatización total del mantenimiento de los programas envoltorio.

□

II.5.4.1.1.2.3. Herramientas gráficas para la generación supervisada de envoltorios

Este enfoque, al igual que el primero descrito, se basa en la construcción de algún tipo de lenguaje que facilite la definición de programas de extracción de datos especializados para cada fuente de información. La diferencia primordial es que, en grado total o parcial, las complejidades del uso de dicho lenguaje son transparentes al creador del envoltorio, que simplemente utiliza algún tipo de herramienta gráfica interactiva que le guía a través de la construcción del programa envoltorio. El objetivo es mantener la capacidad expresiva del enfoque de lenguajes de definición, que le habilita para tratar fuentes complejas, pero consiguiendo mediante el uso de la herramienta gráfica, que su facilidad de uso sea comparable a la de los sistemas de aprendizaje inductivo, evitando así que los creadores de envoltorios deban tener capacidades de programación. Para conseguir esta última funcionalidad, en ocasiones la herramienta supervisora puede incluir algunas heurísticas y técnicas simples de aprendizaje.

En XWRAP[LPH00] el generador de envoltorios interactúa con una herramienta gráfica que le permite definir reglas de extracción de información. Dicha herramienta es capaz además de utilizar heurísticas para tratar de "adivinar" la estructura de ciertas páginas y también de aplicar algoritmos de aprendizaje para detectar patrones repetitivos.

Más concretamente, el proceso de generación de envoltorios en XWRAP puede dividirse en cuatro fases (cada una de las cuales consta a su vez de diversas etapas):

1. Normalización sintáctica. En esta fase: 1) se descarga una página de ejemplo de la fuente de la que se extraerán las tuplas, 2) se corrigen posibles errores sintácticos del HTML obtenido (algo frecuente en fuentes web reales), y 3) se construye un árbol sintáctico de la página.
2. Extracción de información. En esta fase, a través de heurísticas y de la interacción con el usuario a través de una herramienta gráfica, se generan las reglas de extracción que actuarán sobre el árbol para obtener las tuplas a extraer. Este proceso se realiza en tres etapas: 1) identificación de regiones relevantes de la página, 2) identificación de la posición en el árbol de los

elementos relevantes a extraer y 3) identificación de las estructuras jerárquicas que relacionan los datos extraídos.

3. Generación de código. Partiendo de las reglas obtenidas en el paso 2), se genera un código JAVA capaz de implementar las reglas previamente generadas.
4. Prueba. Para poder depurar el envoltorio, el usuario introducirá una serie de URLs alternativas de la misma fuente para comprobar si las reglas de extracción se comportan de manera correcta también con los nuevos ejemplos. Si no fuese así, el usuario puede cambiar las reglas de extracción de manera interactiva, hasta que finalmente se obtenga un conjunto de reglas capaz de tratar adecuadamente todos los ejemplos.

En la fase 1), la etapa de descarga de la página se realiza con la ayuda de un simple cliente http, por lo que se mantienen todos los problemas de este enfoque mencionados en el apartado II.5.4.1.1.1. La etapa de corrección sintáctica corrige errores habituales en páginas reales como ausencia de marcas de fin, marcas vacías o inútiles, o marcas anidadas de manera incorrecta. En la última etapa se utiliza un sistema muy similar a un típico analizador sintáctico de HTML que construye un árbol en el que las marcas HTML conforman los nodos internos y los textos visibles son las hojas del mismo. Además, XWRAP define una serie de funciones de manipulación de los nodos del árbol que permiten obtener su tipo, su nombre y su camino hasta la raíz.

En la fase 2), la etapa 1), de identificación de regiones relevantes la realiza el usuario marcando gráficamente regiones del árbol. Entonces, para cada tipo de región, XWRAP identifica todas las subregiones y define reglas de extracción de regiones para ellas, según su tipo (los tipos de región incluyen tablas, listas, áreas de texto, y otras construcciones HTML. Cada tipo tiene asignado un conjunto de reglas para tratar sus peculiaridades)

La etapa de identificación de los elementos a extraer (etapa 2 de la fase 2) funciona de la siguiente manera: El usuario selecciona en el árbol los diferentes campos de una tupla que vayan a extraerse, los etiqueta y dice cómo se componen entre sí para formar una tupla válida. Entonces, el sistema aplica algoritmos y heurísticas simples de aprendizaje para tratar de encontrar otros ejemplos de tuplas contenidos en la región: por ejemplo, si se han marcado las diferentes columnas de una fila de una tabla, puede inferir que cada fila de la tabla es una tupla válida. Si el sistema infiere algo inválido, el usuario puede corregirle hasta que las reglas definidas sean correctas.

Por último, la etapa de identificación de estructuras jerárquicas se realiza de una manera interactiva similar. XWRAP utiliza heurísticas para inferir una posible organización jerárquica de las regiones de una página y el usuario puede realizar correcciones. Las heurísticas utilizadas por XWRAP utilizan, por un lado, la estructura de las marcas HTML (por ejemplo, si una región es una subtabla de otra, probablemente las tuplas de la región interna sean subrelaciones de las tuplas de la región externa). Por otro lado, para identificar jerarquías dentro de porciones de texto, utiliza heurísticas como suponer que las cabeceras de una jerarquía estarán escritas con una fuente de tamaño mayor que los elementos de niveles inferiores. Estas heurísticas

fallarán con frecuencia y, en ese caso, es el usuario quien hace las correcciones oportunas.

El proceso interactivo visto para la fase 2) es también el que se utiliza en la fase de Prueba, hasta conseguir un conjunto de reglas que traten correctamente todos los ejemplos.

En [BLP01], se presenta una extensión de XWRAP que afirma poder prescindir de la operación semi-automática, pasando el proceso de generación de envoltorios a ser automático. En este caso, para la identificación de regiones relevantes se utilizan un conjunto de heurísticas simples basadas en suponer que los datos relevantes estarán en aquella región en la que haya mayor cantidad de datos visibles. Para la extracción de información relevante, el sistema trata de identificar patrones que se repiten frecuentemente en la página y asume que ese es el patrón de las tuplas a extraer. Para inferir los delimitadores entre los atributos de una tupla, se usan un conjunto de separadores frecuentes (tales como comas o puntos y comas) y algunas heurísticas simples. Finalmente, el usuario etiqueta los diferentes campos del patrón de una tupla. En realidad, estas heurísticas son demasiado simples para tratar muchas fuentes y, en la práctica, funcionarán bien sólo con un tipo específico de ellas. Por ello, el "nuevo" sistema XWRAP debe ser visto más como un complemento útil del anterior que puede automatizar el proceso para fuentes simples, que como un sustituto del mismo.

La valoración que se puede hacer de XWRAP es que es un sistema mixto que trata de retener las principales ventajas de los sistemas de aprendizaje sin perder la potencia de los enfoques basados en lenguajes de especificación. En ese sentido, es un enfoque interesante ya que, para fuentes sencillas se requerirá poca intervención humana (ya que las heurísticas y algoritmos de XWRAP serán efectivos) y, para fuentes complejas, las reglas pueden ser corregidas interactiva o incluso manualmente por el usuario (aunque el lenguaje de especificación de reglas de extracción es de bastante bajo nivel en este caso).

Sin embargo, XWRAP adolece también de limitaciones:

- Las complejidades del establecimiento de sesiones y tratamiento de Javascript en fuentes deben ser manejadas manualmente.
- Sólo es capaz de extraer tuplas de una sola página. No es capaz de extraer tuplas que tengan sus atributos dispuestos en más de una página (algo que ocurre muy frecuentemente, por ejemplo si para ver ciertos atributos de una tupla hay que recorrer un enlace) ni listas de tuplas distribuidas a lo largo de varias páginas. Para ello, sería necesario definir varios envoltorios y combinar manualmente sus resultados.
- Al basar su funcionamiento en la construcción inicial de un árbol, el sistema no puede funcionar de manera asíncrona (precisa que la página se reciba totalmente antes de comenzar la extracción)
- Las heurísticas utilizadas para generación automática son, en ocasiones, muy poco generales, por lo que su utilidad está restringida a pocos casos.

- Aparentemente, no es posible extraer datos de los atributos de las marcas HTML. Los valores de algunos de estos atributos (tales como los 'href' de los enlaces, o los 'src' de las imágenes) son útiles para diversas aplicaciones y no podrían extraerse con XWRAP.

□

Lixto [BFG01] proporciona una herramienta gráfica interactiva que permite a sus usuarios definir gráficamente *elementos de información* a ser extraídos desde una determinada página HTML. Estos elementos pueden estar relacionados entre sí jerárquicamente, proporcionando así soporte para relaciones anidadas.

Cada *elemento de información* se define a través de uno o más *filtros*. Un filtro se expresa mediante una expresión que permite identificar una serie de nodos similares dentro del árbol de una página HTML indicando un camino desde la raíz del árbol. Por ejemplo, la expresión `'.*.table.*.tr'` selecciona todas las filas (nodos <TR>) de todas las tablas de la página.

Como puede verse en el ejemplo, el carácter '*' actúa como comodín en las rutas, por lo que la porción de la ruta `'.*.table'` identifica todas las tablas de la página, y `'.*.table.*.tr'` todas sus filas.

Cada filtro puede tener asociadas una serie de *condiciones* restrictivas que restrinjan los nodos seleccionados por la expresión. Estas condiciones pueden ser de cuatro tipos:

- Condiciones *antes / después*. Indican que la instancia seleccionada por el filtro debe estar antes o después de un determinado elemento.
- Condiciones *no antes / no después*. Indican que la instancia seleccionada por el filtro *no* debe estar antes o *no* debe estar después de un determinado elemento.
- Condiciones *internas*. Indican que un determinado sub-elemento *no* debe aparecer dentro de los elementos seleccionados por el filtro.
- Condiciones de *rango*. Permite seleccionar un sub-rango dentro de los elementos seleccionados.

De esta manera, las instancias de datos extraídas para cada *elemento de información* serán aquellas que concuerden con al menos uno de sus *filtros* asociados y satisfagan todas las *condiciones* asociadas a dicho filtro.

El proceso iterativo seguido por el usuario para definir las acciones de extracción de datos para cada uno de estos elementos de información es el siguiente:

1. El usuario selecciona directamente en la página de su navegador una instancia del elemento de información.

2. Entonces Lixto genera automáticamente un filtro que selecciona elementos similares al seleccionado (e.g. si el usuario marca una fila de una tabla, Lixto puede marcar todas las filas de tablas presentes en la página). S
3. Si Lixto ha seleccionado más elementos de los precisos, el usuario puede aplicar condiciones para eliminar los no deseados.
4. Si Lixto ha seleccionado menos elementos de los deseados, el usuario marcará nuevamente en su navegador uno de los ejemplos de instancia del elemento de información que *no* hayan sido seleccionados por Lixto, de manera que se generará un nuevo filtro.
5. El proceso termina cuando Lixto selecciona exactamente las instancias deseadas.

Una vez realizado este proceso, Lixto transforma los filtros y condiciones definidos gráficamente por el usuario en expresiones de un lenguaje de definición de programas de extracción de datos llamado *Elog*. Sin embargo, tal y como se ha visto, el usuario no precisa ni siquiera ser consciente de la existencia del lenguaje, ya que este está *oculto* tras la interfaz proporcionada por la herramienta gráfica.

Lixto consigue conjugar capacidad expresiva y facilidad de uso, lo cuál le convierte en un sistema atractivo.

Entre sus inconvenientes, se encuentra el que el proceso de creación de envoltorios será bastante tedioso en la mayoría de las fuentes, ya que puede ser necesario definir un número alto de filtros y condiciones para cada elemento de información y, además, es necesario hacerlo incluso para los elementos de información atómicos. Por ejemplo, si se desea construir un envoltorio para una tienda electrónica de libros, que extraiga para cada libro información sobre su título, autor, precio y editorial, deberían definirse 5 elementos de información: uno no atómico (`LIBRO`) y 4 atómicos que serían subelementos del anterior (`TITULO`, `AUTOR`, `EDITORIAL` y `PRECIO`). Para cada uno de esos elementos sería necesario, probablemente, construir varios filtros y condiciones, por lo cuál, incluso para este caso tan sencillo, el proceso de creación del envoltorio puede ser largo.

Otro inconveniente es que, al igual que los sistemas previos, tampoco ofrece funcionalidades para tratar con las dificultades asociadas a la navegación a través de fuentes web complejas.

□

II.5.4.1.1.2.4. Comparación de los diversos enfoques para extracción de tuplas en fuentes web

No es fácil establecer comparaciones a nivel global entre los diferentes enfoques vistos, ya que hay diferencias significativas incluso dentro de las técnicas del mismo tipo.

En general, los sistemas basados en lenguajes de definición se muestran efectivos incluso con fuentes muy complejas, pero la construcción de programas de extracción suele estar sólo al alcance de expertos, incluso para las fuentes más simples.

En cambio, las técnicas basadas en aprendizaje inductivo tienen la ventaja de requerir menor intervención humana, y cuando la requieren, menos especializada, por lo que su coste asociado es menor. Algunos sistemas pueden incluso generar y mantener ciertos tipos simples de envoltorios de manera totalmente automática. Por el contrario, este tipo de técnicas suelen tener una capacidad expresiva muy limitada en sus capacidades de extracción, lo que las inhabilita para tratar adecuadamente con muchas fuentes web reales. Otro inconveniente importante es que normalmente es necesario proporcionar un número de ejemplos bastante elevado para que el algoritmo inductivo alcance la convergencia.

El enfoque de herramientas gráficas "supervisoras" pretende recoger las ventajas de los dos enfoques anteriores y parece la mejor posibilidad, hoy en día, para un sistema que aspire a ser utilizado industrialmente. Las razones que sustentan esta afirmación son las siguientes:

- En un sistema que aspire a ser completo y funcionar con cualquier tipo de fuentes, es necesario poder tratar fuentes complejas. Eso descarta a los sistemas puros de aprendizaje inductivo actuales (si bien esto no excluye la posibilidad de usar estas técnicas como complemento en aras de lograr un grado de automatización mayor para las fuentes más simples).
- Tal y como se afirma en el reciente informe de experiencia industrial en integración de datos [SH01], las herramientas de creación de envoltorios para fuentes web deberían orientarse a usuarios sin capacidades técnicas. Esto excluye los enfoques puros de lenguajes de definición.

Hasta el momento, el enfoque híbrido proporcionado por las herramientas gráficas supervisoras es el único que puede satisfacer ambas necesidades a la vez.

El presente trabajo realiza en este campo una de sus principales aportaciones al presentar un nuevo sistema de generación semi-automática de programas envoltorio que sigue en líneas generales el enfoque de herramienta supervisora y, además, como ya se ha comentado soluciona el problema planteado por las secuencias complejas de navegación que involucran Javascript, técnicas de mantenimiento de sesión, etc. Este sistema es descrito en el apartado III.4.

Finalmente, hay que mencionar que la actual emergencia de XML podría permitir a los constructores de sitios web exportar sus datos en un formato más estructurado, facilitando así en gran medida la construcción de programas envoltorio. Sin embargo, en la actualidad, pocas fuentes web han adoptado este enfoque y no se perciben movimientos claros en este sentido, al menos para fuentes comerciales, con lo que parece claro que las técnicas de extracción desde páginas HTML serán necesarias durante bastante tiempo. Además, incluso cuando las fuentes exporten sus datos en

XML, existirán aún problemas derivados de las heterogeneidades de esquema, formatos y taxonomías entre las fuentes, por lo que todo hace pensar que las técnicas para generación de envoltorios seguirán siendo necesarias (aunque puedan ser distintas a las actuales).

III. DESCRIPCIÓN DEL SISTEMA MEDIADOR

En este capítulo se presenta el sistema mediador objeto de este trabajo. Se comienza delineando su arquitectura global. Posteriormente se analizan en detalle los componentes de la arquitectura haciendo especial hincapié en su nivel lógico y en el sistema de generación semi-automática de envoltorios para fuentes web y de texto semi-estructurado.

III.1. DESCRIPCIÓN DE LA ARQUITECTURA

En este apartado se presenta la arquitectura del sistema. Esta arquitectura es genérica y podría ser utilizada por otros sistemas basados en el concepto de "mediador", si bien ha sido concebida con la idea de dar soporte, fundamentalmente, al sistema que ocupa el presente trabajo.

III.1.1. OBJETIVOS DE LA ARQUITECTURA

El objetivo fundamental de la arquitectura es servir de base y marco de trabajo para la elaboración de Sistemas Mediadores que puedan ser utilizados por la industria para crear y mantener aplicaciones de integración de datos estructurados y semi-estructurados.

El foco de la arquitectura se centra en satisfacer las necesidades de integración de información heterogénea planteadas por el entorno empresarial, permitiendo la construcción de aplicaciones fácilmente *integrables* en dicho entorno, con la adecuada *eficiencia* y además con la capacidad de conseguir de manera cómoda *escalabilidad* y *extensibilidad* en las aplicaciones construidas apoyándose en ella. Todas estas características son imprescindibles en un sistema que aspire a ser utilizado en el moderno entorno de negocio.

Para favorecer su *integración* en los diferentes entornos de producción empresariales y acortar los tiempos de formación, la arquitectura debe apoyarse en normas y conductas que cuenten con el apoyo de la industria. Ejemplos de normas relevantes apoyadas por la industria pueden ser el lenguaje de consulta SQL[SQL92] o la interfaz de conexión a Bases de Datos JDBC[JDB01].

Para favorecer la *extensibilidad*, y especialmente dentro del ámbito de tecnologías incipientes como es ésta, es necesario utilizar en su construcción un enfoque orientado a componentes, que permita añadir, suprimir o reemplazar con facilidad elementos de la arquitectura, a medida que nuevas técnicas vayan siendo desarrolladas y que nuevas necesidades vayan siendo detectadas.

Para favorecer la *eficiencia* de las aplicaciones realizadas, es necesario disponer de módulos para la optimización de consultas, mantenimiento de caches, precarga de datos, etc.

Para incrementar la *escalabilidad* de las aplicaciones construidas basándose en la arquitectura, es necesario apostar por un enfoque *distribuido* que permita colocar los componentes en diferentes máquinas, replicarlos, etc.

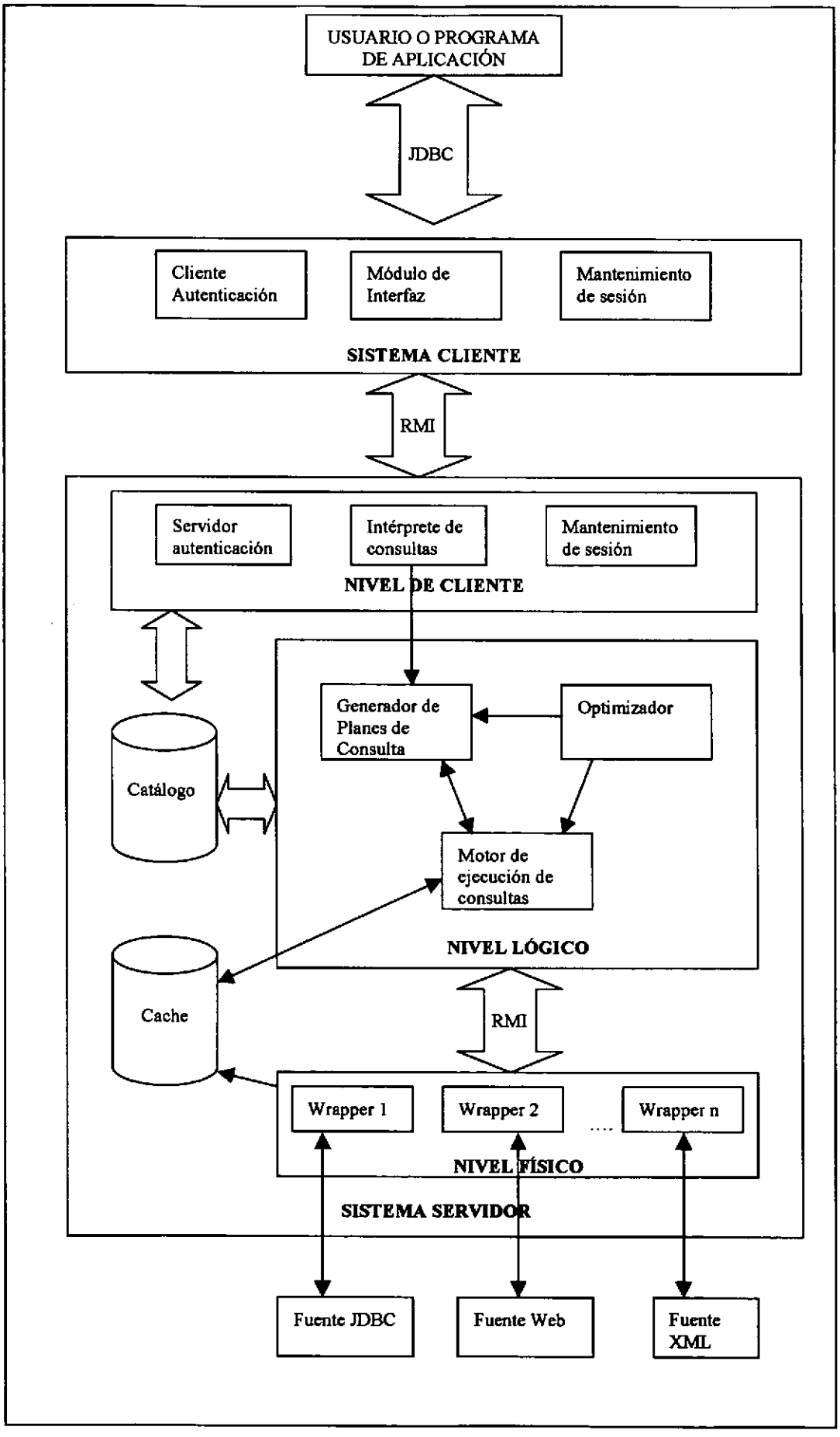
III.1.2. DESCRIPCIÓN GENERAL

La Figura III.1.2.1 muestra una visión general de la arquitectura.

Una primera consideración es que la arquitectura se ha diseñado teniendo en mente la arquitectura de desarrollo de aplicaciones distribuidas basada en el lenguaje JAVA[JAV00], que se está convirtiendo en una norma "*de facto*" en la industria para el desarrollo de aplicaciones orientadas a objetos distribuidas y multi-plataforma.

En la figura se muestran un sistema cliente y un sistema servidor.

Una aplicación de usuario puede establecer una *sesión* o *conexión* con un sistema cliente a través de la norma JDBC (Java DataBase Connectivity)[JDB01], especificando una URL al servidor al que desea conectarse, un identificador de usuario y una contraseña.


USUARIO O PROGRAMA DE APLICACIÓN
JDBC
Cliente Autenticación
Módulo de Interfaz
Mantenimiento de sesión
SISTEMA CLIENTE
RMI
Servidor autenticación
Intérprete de consultas
Mantenimiento de sesión
NIVEL DE CLIENTE
Generador de Planes de Consulta
Optimizador
Catálogo
Motor de ejecución de consultas
NIVEL LÓGICO
Cache
RMI
Wrapper 1
Wrapper 2
....
Wrapper n
NIVEL FÍSICO
SISTEMA SERVIDOR
Fuente JDBC
Fuente Web
Fuente XML

Un sistema cliente puede establecer *sesiones* con uno o varios sistemas servidores, ejecutando el proceso de autenticación en las mismas, con el identificador de usuario y la contraseña recibidos a través de la petición de establecimiento de conexión JDBC.

El sistema cliente y el sistema servidor pueden estar en la misma máquina o en máquinas diferentes. La comunicación entre ellos se realiza a través de la interfaz de programación Java RMI[RMI01], también ampliamente utilizada en la industria.

Opcionalmente, las comunicaciones entre el sistema cliente y el servidor pueden ir cifradas y autenticadas. La implementación de esta funcionalidad se puede realizar de manera sencilla haciendo que Java RMI trabaje con una factoría de sockets seguros según la norma SSL[SSL96]. Existen en el mercado numerosos productos que implementan factorías de sockets seguros SSL y que son directamente integrables con Java RMI.

En el sistema servidor, el *nivel de cliente* se ocupa de establecer y mantener la sesión con el cliente. También es responsabilidad suya aceptar las consultas en SQL, traducirlas a álgebra relacional enviándoselas al *nivel lógico*, y devolver las respuestas obtenidas por dicho nivel al sistema cliente.

El *nivel lógico*, por su parte, se ocupa de comprobar si la consulta es *factible* (es decir, si las capacidades de las fuentes permiten contestarla), de generar planes de ejecución para la consulta, escoger el más óptimo, y resolver la consulta interactuando con el *nivel físico*. Las comunicaciones entre *nivel lógico* y *nivel físico* se realizan nuevamente a través de la interfaz de programación Java RMI. Esto permite que el nivel lógico pueda estar en una máquina diferente de aquella o aquellas en las que se encuentra el *nivel físico*.

El *nivel físico* tiene como misión conseguir que las fuentes finales de datos se comporten de cara a los niveles superiores de la arquitectura de acuerdo a un modelo común de relación. Esta labor es realizada por los *programas envoltorio* o *wrappers*. Dado que las comunicaciones con el nivel lógico se realizan a través de Java RMI, cada *envoltorio* o *wrapper* puede estar en una máquina distinta, lo que permite al sistema escalar fácilmente a cualquier número de fuentes. Otro aspecto importante es que a través de un *envoltorio*, un mediador puede comunicarse transparentemente con otros mediadores, por lo que es posible edificar nuevos mediadores sobre otros ya existentes.

Todos los niveles precisan para su funcionamiento de la consulta/actualización del *catálogo* o *diccionario de datos*.

A continuación, se repasan los principales componentes de la arquitectura y se proporciona una breve descripción de los mismos.

III.1.2.1. NIVEL FÍSICO

La misión de este nivel es conseguir que las fuentes finales de datos se comporten de cara a los niveles superiores de la arquitectura como relaciones con una estructura definida, que exportan a los niveles superiores de la arquitectura. Estas labores son realizadas a través de los *programas envoltorio* o *wrappers*. Como ya se ha comentado, las comunicaciones con el nivel lógico se realizan a través de Java RMI.

Las tareas principales que debe asumir un *programa envoltorio* en la arquitectura son:

- Exportar a los niveles superiores las capacidades de consulta soportadas por la relación.
- Mantener y exportar estadísticas de costes para las consultas efectúadas sobre la relación.
- Traducir las subconsultas que le envíe el nivel superior desde el formato del mediador al formato nativo de la fuente.
- Convertir la salida devuelta por la fuente en tuplas de la relación exportada.

Obviamente, el aspecto de generación de *programas envoltorio* es crucial en la arquitectura. Es fundamental disponer de herramientas que permitan generar rápidamente y mantener con facilidad los *programas envoltorio*, al menos para los tipos de fuentes más comunes que son, fundamentalmente: fuentes web, fuentes JDBC, fuentes XML y otros mediadores.

III.1.2.2. NIVEL LÓGICO

Las responsabilidades del nivel lógico se centran en integrar y combinar las relaciones exportadas por los diferentes *programas envoltorio* para componer el esquema global del mediador.

Esta combinación se realiza mediante la definición de vistas, expresadas en álgebra relacional, sobre las relaciones exportadas sobre el nivel físico (lo cual se corresponde con el enfoque GAV, tal y como se describió en el apartado II.5.1.1). En la definición de vistas podrán utilizarse uniones, joins, selecciones y proyecciones, además de operaciones de agregación/ordenación y algunas construcciones específicas para tratar la heterogeneidad de formatos (ver apartado III.2.6).

El nivel lógico es el que recibe las consultas realizadas sobre el esquema global expresadas en álgebra relacional, comprueba si son resolubles en función de las capacidades de consulta de las fuentes, elabora los posibles planes de ejecución de la consulta, escoge el más óptimo y ejecuta dicha consulta devolviendo a la capa superior los resultados obtenidos. Dentro de los procesos de optimización, es responsabilidad de este nivel acceder a las fuentes en paralelo siempre que sea posible, así como realizar la gestión de dicho paralelismo.

También es responsabilidad de este nivel el tratamiento de ciertas heterogeneidades de representación y de esquema que pueda haber en las distintas relaciones exportadas por los *programas envoltorio*.

Internamente, el nivel lógico consta de tres módulos fundamentales:

- El *Generador de Planes de Consulta*. Su misión es decidir si la consulta recibida puede o no ser contestada, de acuerdo a las capacidades de consulta soportadas. En caso afirmativo, debe generar todos los posibles planes de ejecución (que aquí llamaremos *métodos de búsqueda*) para ella.
- El *Optimizador* tiene como misión calcular el coste de cada plan de ejecución y escoger el más óptimo de entre todos los posibles.
- El *Motor de Ejecución* recibe un plan de ejecución para la consulta y lo realiza obteniendo los resultados de la misma. Esta tarea involucra las tareas de interactuar con el nivel físico, gestionar el paralelismo en el acceso a las fuentes, acceder a la cache cuando sea posible y, finalmente, integrar los resultados obtenidos de cada fuente.

Durante todas las operaciones del nivel lógico es constante el acceso al catálogo, donde está almacenada información tal como las capacidades de consulta de las vistas creadas en el nivel lógico, los esquemas de las relaciones, la información de costes, etc.

III.1.2.3. NIVEL DE CLIENTE

Este nivel se ocupa de realizar la interfaz entre el sistema cliente y el sistema mediador. Es, por lo tanto, responsabilidad suya proporcionar los medios para que un cliente pueda autenticarse y establecer una sesión con una base de datos, enviarle consultas y obtener los resultados correspondientes a las mismas.

Las consultas son recibidas en el *lenguaje de consultas externo* del mediador, que es similar al lenguaje SQL. El intérprete de consultas las traduce a álgebra relacional entes de entregársela al nivel lógico.

Cuando el nivel lógico devuelve una respuesta para la consulta efectuada, es también responsabilidad de este nivel proporcionársela al sistema cliente que la envió.

III.1.2.4. SISTEMA DE CACHE

El sistema de cache tiene como misión almacenar los resultados de consultas previas efectuadas sobre el sistema, de manera que posteriores consultas realizadas sobre él mismo puedan contestarse, ya sea en parte o en su totalidad, sin necesidad de acceder de nuevo a las fuentes.

La cache es mantenida a nivel de relación. Esto quiere decir que puede haber una tabla en la base de datos JDBC por cada relación. La cache puede configurarse para funcionar *a nivel de relación base* (lo que quiere decir que sólo se guarda cache de las relaciones base), a *nivel global* (con lo que sólo se guardaría cache de las relaciones del esquema global) o a *nivel total* (se guarda cache de todas las relaciones, incluidas las intermedias de los árboles de vistas). Los dos primeros modos pueden funcionar a la vez y esa es, quizás, la configuración más común.

A medida que se vayan realizando consultas, las tablas se rellenan con las tuplas obtenidas. De esta manera, consulta a consulta, se puede ir construyendo una réplica parcial en el entorno local de las relaciones base de las fuentes. Obviamente, para que el mediador pueda saber qué consultas sobre la relación son solubles con datos de la cache y cuales no, cada una de estas replicas debe mantener una descripción de sus contenidos actuales. La manera en que una relación puede describir sus contenidos para integrarse con el algoritmo de resolución de consultas es similar al enfoque LAV (ver apartado II.5.1.2) y se describe con detalle en el apartado III.2.5.6.

III.1.2.4.1. PLANIFICADOR DE ACTUALIZACIONES

Si bien no está impuesto por la arquitectura, el sistema incluye una utilidad que permite planificar temporalmente tareas de actualización de datos. Una tarea consiste en un conjunto de consultas y acciones sobre una base de datos del sistema, a realizar en un determinado día y hora. También pueden definirse horas de inicio y de fin entre las que ejecutar una tarea a intervalos periódicos.

Por defecto, la salida de estas consultas alimentará la cache de la base de datos sobre la que se realicen, con lo cual es una manera de "pre-cargar" en cache ciertas consultas o incluso todos los datos de las relaciones base. De esta manera, el sistema mediador puede funcionar, ya desde su puesta en marcha, total o parcialmente en modo *almacén de datos* (o *warehouse*).

Para procesar la salida de estas consultas de otras maneras que puedan convenir al programador, se definen los *convertidores de salida*, que permiten tratar los resultados obtenidos de la manera deseada por el programador. Esto puede ser útil, por ejemplo, para extraer información de las fuentes a intervalos periódicos y realizar con ella alguna tarea predeterminada.

Se proporcionan ya realizados convertidores para volcar los resultados directamente a texto plano, XML[XML00] y JDBC.

Se permite al programador la definición de nuevos convertidores mediante la implementación de una interfaz JAVA.

III.1.2.5. CATÁLOGO O DICCIONARIO DE DATOS

El diccionario de datos de un mediador debe presentar una estructura diferente de la de un diccionario de datos tradicional. Tiene que ser extendido para almacenar las

definiciones del esquema mediado, las capacidades de las fuentes, información de costes, reglas para tratar la heterogeneidad de formatos en las fuentes, etc.

La organización básica del catálogo es la siguiente:

- El diccionario de datos está organizado en distintas 'Bases de Datos'.
- Cada Base de datos tiene un esquema global.
- Diversos usuarios tienen permisos diferentes sobre los esquemas. Los usuarios deben autenticarse según un esquema 'identificador de usuario-contraseña'.
- El esquema de una Base de Datos está compuesto por un conjunto de relaciones "virtuales".
- Cada relación del esquema puede o bien ser una relación atómica o *relación base* (que son aquellas relaciones exportadas por las fuentes) o venir definida por una *expresión en álgebra relacional*, expresada en función de otras relaciones.
- Para cada relación (ya sea atómica o no) se almacenarán sus *capacidades de consulta*. En nuestro sistema las capacidades de consulta se representan mediante *métodos de búsqueda*, que conceptualmente son las diferentes maneras en las que la relación permite ser consultada.
- Para cada relación se almacenará su *información de costes*, que se utilizará para propósitos de optimización.
- Se almacenará también para cada relación, las posibles reglas que permitan tratar con heterogeneidades de esquema o formato y que sea necesario realizar para que la fuente sea integrable en el esquema global.
- Para cada relación base, para cada método de búsqueda, se almacenará:
  - Información sobre la manera de implementar ese método de búsqueda en la fuente. Es decir, ¿Cómo traducir una subconsulta enviada por el mediador, y que puede ser contestada por ese método de búsqueda, al formato de la fuente?.
  - Información sobre como "entender" los resultados ofrecidos por el método de búsqueda en la fuente, de manera que puedan extraerse las tuplas que constituyen el resultado a la consulta. Esto normalmente involucrará almacenar una especificación escrita en un lenguaje especial que permita al programa envoltorio realizar un proceso de extracción por el que obtendrá las tuplas deseadas.

En apartados subsiguientes se tratará en más detalle la manera concreta en que el sistema propuesto representa en el catálogo este conjunto de *metainformación*. En concreto, los aspectos relacionados con la representación de los esquemas de las relaciones, sus capacidades de consulta, información de costes y tratamiento de heterogeneidad en los esquemas y formatos de representación, se tratan en la descripción detallada del *nivel lógico* que ocupa todo el apartado III.2. La información referente a como implementar un método de búsqueda y como "entender" los resultados devueltos por las fuentes se trata fundamentalmente en los apartado III.3 y III.4.

Aunque no está impuesto por la arquitectura, en el sistema propuesto la información contenida en el catálogo está representada en XML[XML00]. Utilizar XML, que cuenta con un amplio apoyo de la industria, permite sacar partido de herramientas ya construidas a la hora de crear herramientas de acceso al catálogo y de edición de los contenidos del mismo.

III.2. DESCRIPCIÓN DEL NIVEL LÓGICO

El nivel lógico es el núcleo del sistema mediador y, de una u otra manera, involucra a casi todos sus componentes. En este apartado se discutirán en detalle los modelos, técnicas y algoritmos utilizados en su construcción.

En primer lugar, en el apartado III.2.1, se presenta el modelo de relación con el que trabaja el mediador y la manera de crear las *relaciones base* exportadas por las fuentes. También incluye la manera de representar las limitaciones en las capacidades de consulta. La definición de estas relaciones base constituye la interfaz a la que deberán ajustarse los envoltorios en sus comunicaciones con el nivel lógico del mediador.

El apartado III.2.2 presenta el modelo de consulta utilizado por el sistema. El apartado III.2.3 introduce los algoritmos precisos para determinar si una consulta particular es ejecutable por las capacidades de consulta de una relación.

Seguidamente, el apartado III.2.4 explica en detalle el proceso de creación de nuevas relaciones a partir de las relaciones base. Se incluyen aquí todos los algoritmos necesarios para obtener las capacidades de consulta de las relaciones globales, partiendo de las de las relaciones base.

A continuación, en el apartado III.2.5 se muestran los aspectos relativos a la ejecución y optimización de consultas, incluyendo también una descripción del sistema de cache utilizado. Si bien el subsistema de cache no pertenece estrictamente al nivel lógico, sino que es independiente e interactúa tanto con el nivel físico como con el lógico, las técnicas utilizadas en su construcción se asientan sobre las mismas ideas básicas que se expondrán sobre este nivel y, en consecuencia, es en este contexto donde mejor pueden ser comprendidas

Finalmente, el apartado III.2.6 expone los formalismos que subyacen a las técnicas disponibles en el sistema mediador para el tratamiento de heterogeneidades de formatos y taxonomías entre las fuentes.

III.2.1. MODELO DE RELACIÓN

El modelo de relación en el sistema es básicamente relacional, aunque con soporte para atributos de tipo registro y multivaluados (también llamados de tipo array). También es necesario representar la descripción de las capacidades de consulta de la relación.

Así, una relación contiene tuplas que están compuestas por un conjunto de atributos. Cada atributo de una relación pertenecerá a un *tipo de dato*. El tipo de un determinado atributo delimita qué operadores pueden aplicarse sobre él, así como ciertas

restricciones que los elementos de ese tipo deben cumplir. Algunos tipos de datos soportados en la implementación actual del sistema son: cadenas de caracteres, enteros, flotantes, divisas y fechas. Tal y como ya hemos comentado, los tipos de dato registro y array también son permitidos.

Ejemplo III.2.1.1: Definición de una relación

Considérese una tienda electrónica de libros en Internet que, para cada una de las referencias de su catálogo ofrece información sobre su título, su autor, su formato (existen tres posibles: "tapa dura", "tapa blanda" y "bolsillo"), su editorial y su precio.

Podemos modelar esta fuente definiendo en el sistema la siguiente relación:

```
R={TITULO:STRING, AUTOR:STRING, FORMATO:ENUMERATED("tapa dura","tapa
blanda","bolsillo"), EDITORIAL:STRING, PRECIO:MONEY}
```

□

Las relaciones del sistema pueden ser de dos clases: las *relaciones base* que son aquellas exportadas por las fuentes a través de los envoltorios del nivel físico y las *relaciones derivadas*, que son aquellas que se definen mediante algún tipo de vista que combina las *relaciones base*.

III.2.1.1. DESCRIPCIÓN DE LAS CAPACIDADES DE CONSULTA DE LAS RELACIONES

Como ya se ha mencionado, en el contexto en que funciona nuestro sistema, las fuentes y, por lo tanto, las relaciones base) pueden ofrecer capacidades de consulta limitadas (mucho más que las proporcionadas, por ejemplo, por una base de datos convencional). Es necesario, por lo tanto, representar de alguna manera estas restricciones con el fin de que el sistema mediador sepa qué consultas puede ejecutar sobre cada relación.

Cada relación en una fuente representará sus capacidades de consulta mediante lo que llamaremos *métodos de búsqueda*. Si una relación no tuviese ningún método de búsqueda, entonces no podría realizarse ninguna consulta sobre dicha relación.

Cada método de búsqueda está compuesto por tres elementos:

- Capacidades negativas: indican las restricciones que deben ser satisfechas por una consulta efectuada sobre la relación, para que pueda ser ejecutada con este método de búsqueda.
- Capacidades positivas: Indican las capacidades de consulta sobre la relación ofrecidas por este método de búsqueda, una vez que las capacidades negativas han sido satisfechas.
- Conjunto de atributos de salida: Indica el conjunto de atributos que la relación devuelve en la salida de las consultas realizadas a través de este método de búsqueda. Es necesario representar esta información explícitamente ya que algunas

fuentes no devuelven en sus respuestas todos los atributos por los que pueden ser consultadas. Por ejemplo, muchas librerías web permiten consultar su catálogo por 'tema', pero posteriormente no ofrecen esta información en sus páginas de resultados.

Tanto las capacidades positivas como las negativas se representan a través de 4-uplas con el formato (ATRIBUTO, OPERADORES, MULTIPLICIDAD, VALORES_POSIBLES) donde:

- ATRIBUTO es un atributo de la relación.
- OPERADORES es un operador válido para el tipo de dato del atributo. El valor CUALQUIERA representa a cualquier operador
- MULTIPLICIDAD. Indica un número de condiciones de consulta. Puede tomar como valor un número cardinal, o el valor '+' para indicar un valor superior a 0 pero sin límite superior. Las 4-uplas positivas utilizan este campo para indicar cuántas condiciones de selección sobre el atributo y el operador, admite este método de búsqueda en una sola consulta. En las 4-uplas negativas indicará el número mínimo de condiciones de consulta para ese atributo y operador que debe incluir una consulta sobre la relación para poder ser ejecutada con este método de búsqueda.
- VALORES_POSIBLES es la lista de valores por los que puede consultarse el atributo con ese operador, en este método de búsqueda. Si contiene el valor CUALQUIERA significa que el rango de búsqueda no está acotado (dentro del rango asociado al tipo de dato del atributo). y puede consultarse el atributo por cualquier valor.

Ejemplo III.2.1.2: Representación de las capacidades de consulta de una tienda electrónica.

Supongamos que la tienda electrónica modelada por la relación R presenta el formulario de búsqueda que se muestra en la Figura III.2.1.1.

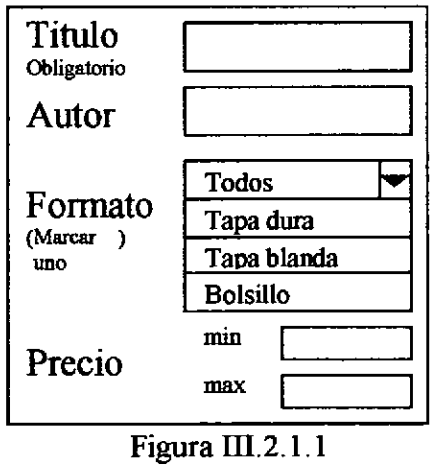


Figura III.2.1.1

El formulario obliga al usuario a especificar al menos un valor para el atributo TITULO y opcionalmente le permite fijar valores para los atributos AUTOR, FORMATO (sólo 3 valores posibles: Bolsillo, Tapa Dura y Tapa Blanda) y PRECIO (para el cual es posible especificar un rango de valores).

Las búsquedas por título y autor son búsquedas por palabra clave (operador CONTIENE). Las cajas de búsqueda para estos atributos pueden rellenarse con un número arbitrario de palabras, y la fuente ejecutará una búsqueda en dicho campo por todas las palabras especificadas, utilizando el operador lógico AND. Por ejemplo, si se rellenase la caja del atributo título con los valores 'java' y 'xml', la fuente devolvería aquellos libros que contuviesen ambas palabras en su título.

Además de los atributos anteriores, la fuente devolverá el atributo EDITORIAL en los resultados de las consultas efectuadas.

Las capacidades de búsqueda de R pueden modelarse mediante el siguiente método de búsqueda:

```
{
  NEGATIVAS {
     (TITULO, CONTIENE, 1, CUALQUIERA)
  }

  POSITIVAS {
    (TITULO, CONTIENE, +, CUALQUIERA)
    (AUTOR, CONTIENE, +, CUALQUIERA)
    (FORMATO, =, 1, {'Tapa Dura', 'Tapa Blanda', 'Bolsillo'})
    (PRECIO,<=, 1, CUALQUIERA)
    (PRECIO,>=, 1, CUALQUIERA)
  }

 SALIDA {TITULO, AUTOR, EDITORIAL, FORMATO, PRECIO}
}
```

No todas las fuentes de datos podrán ser modeladas con un único método de búsqueda. Cuando una relación ofrece posibilidades de consulta alternativas y mutuamente excluyentes, entonces se requiere más de un método de búsqueda. A continuación se ofrecen dos posibles variaciones de este ejemplo que harían preciso definir más de un método de búsqueda:

1) Si la fuente, en lugar de hacer obligatoria la búsqueda por TITULO, obligase a buscar o bien por TITULO o bien por AUTOR (sólo uno de ellos es requerido), entonces sería necesario añadir un segundo método de búsqueda igual al anterior con el único cambio de que, en la sección de capacidades negativas se sustituiría la 4-upla para el atributo TITULO por otra para el atributo AUTOR de la forma (AUTOR, CONTIENE, 1, CUALQUIERA).

2) Si el formulario de consulta añadiese una casilla de verificación que, cuando estuviese marcada, hiciese que las búsquedas por `TITULO` y `AUTOR` pasasen de utilizar el operador `CONTIENE` a utilizar el operador '=' (búsqueda exacta). En ese caso, sería necesario añadir un segundo método de búsqueda en el cual las 4-uplas asociadas a dichos atributos llevarían este último operador en lugar del primero.

□

III.2.1.1.1. REDUNDANCIA DE 4-UPLAS

La lista de 4-uplas para un atributo en las capacidades positivas o negativas de un método de búsqueda puede contener elementos redundantes, que pueden ser eliminados para obtener una representación más compacta y eficiente. Esto será de especial importancia durante el proceso de generación automática de los métodos de búsqueda de las relaciones derivadas.

En concreto:

1) Una 4-upla en la lista de capacidades negativas es redundante (y, por lo tanto, puede ser eliminada) si *es menos restrictiva* que alguna otra 4-upla de la lista. Una 4-upla (`A, OP1, MUL1, VALORES1`) es menos restrictiva que otra 4-upla (`A, OP2, MUL2, VALORES2`) si se cumplen las siguientes condiciones:

- `OP1=OP2` (Se considera que `CUALQUIERA` es igual a `CUALQUIERA`).
- `MUL1 >= MUL2` ('+' es considerado como el mayor valor posible)
- `VALORES2` $\subseteq$ `VALORES1` (`CUALQUIERA` es el conjunto de todos los valores permitidos por el tipo de dato de `A`).

2) Una 4-upla de la lista de capacidades positivas es redundante si existe alguna otra 4-upla que permita la ejecución de todas las consultas que permite la primera. Asumiendo que todas las 4-uplas para el mismo atributo, operador y conjunto de valores están "fusionadas" en una sola 4-upla con la suma de sus multiplicidades, una 4-upla (`A, OP1, MUL1, VALORES1`) puede eliminarse si existe otra 4-upla (`A, OP2, MUL2, VALORES2`), donde:

- `OP1=OP2` (`CUALQUIERA` es un "comodín" que puede referirse a cualquiera de los operadores permitidos por el tipo de dato de `A`).
- `MUL1 <= MUL2`
- `VALORES1` $\subseteq$ `VALORES2` (`CUALQUIERA` es el conjunto de todos los operadores permitidos por el tipo de dato de `A`).

III.2.2. MODELO DE CONSULTA

Una *consulta base* en nuestro sistema es de la forma (utilizando notación SQL) :

```
Select * from R where (CondicionConsulta1) and … and
(CondicionConsultaN)
```

Dónde una *Condición de Consulta* (o *Condición de Selección*) es una expresión booleana que combina 3-uplas de la forma `(Atributo operador valor)`.

Ejemplo III.2.2.1: Una consulta base sobre R.

```
select * from R where (titulo contiene 'java')and(titulo contiene
'xml') and (precio <= 5000) and (editorial contiene 'Addison-Wesley')
```

Es importante notar que una consulta base no admite que las condiciones de consulta de primer nivel sean combinadas mediante otro operador booleano diferente de AND. Por ejemplo la consulta:

```
select * from R where (titulo contiene 'java') or (editorial contiene
'McGraw-Hill')
```

no sería una consulta base válida. El sistema sería capaz de contestarla, pero descomponiéndola en las siguientes dos sub-consultas base:

```
select * from R where (titulo contiene 'java')
select * from R where (editorial contiene 'McGraw-Hill')'
```

y realizando posteriormente la unión de ambos sub-resultados.

El sistema sí podrá ejecutar en una sola consulta condiciones booleanas que no sean de primer nivel. Por ejemplo:

```
select * from R where (titulo contiene 'java')and(titulo contiene
'xml') and (precio <= 5000) and ((editorial contiene 'Addison-
Wesley') or (editorial contiene 'McGraw-Hill'))
```

pero estas condiciones de consulta serán ejecutadas mediante post-procesados. Es decir, se efectuará primero sobre la relación la consulta:

```
select * from R where (titulo contiene 'java')and(titulo contiene
'xml') and (precio <= 5000)
```

y entonces el mediador filtrará los resultados obtenidos para descartar aquellos que no cumplan la condición adicional:

```
((editorial contiene 'Addison-Wesley') or (editorial contiene
'McGraw-Hill'))
```

Esto sólo puede hacerse con condiciones que afecten a atributos que estén presentes en el conjunto de atributos de salida. Si no fuese así, el mediador no podría,

obviamente, realizar el post-procesado. En el apartado III.2.3.1 se comentarán con más detalle las técnicas de post-procesado.

□

Pueden realizarse consultas sobre más de una relación utilizando para ello los operadores unión y join. También se soportan otras operaciones de álgebra relacional, como proyecciones, selecciones y agregaciones. En concreto las operaciones de agregación disponibles incluyen operaciones de agrupamiento (equivalente al `GROUPBY` de SQL), conteo y operaciones aritméticas (`SUM`, `COUNT`, `MIN`, `MAX`, etc.)

En este punto es interesante realizar una comparación entre nuestro modelo de consulta y el de *consultas conjuntivas* (sin soporte para predicados adicionales), que fue ya comentado en la sección II.5.1.2.1, y que es la base de los modelos de consulta de muchos de los sistemas mediadores presentados hasta el momento [GMPQ95][LRO96] [YLGU99].

La principal diferencia con nuestro modelo estriba en que en las condiciones de consulta sólo puede utilizarse el operador '=' y, además, sólo se permite una condición de consulta por atributo. Por lo tanto, no puede representar adecuadamente las capacidades de muchas fuentes que nuestro sistema sí puede. Así en la fuente del ejemplo III.2.1.2:

- No es posible representar las capacidades de consulta para el atributo `PRECIO`, ya que la fuente utiliza para ello los operadores '<=' y '>=' en lugar de '='.
- No es posible representar adecuadamente las capacidades de consulta de los atributos `TITULO` y `AUTOR`, ya que ambos utilizan el operador `CONTIENE` (que sólo puede ser emulado de forma pobre por el operador '='). Además, la fuente permite que se especifique más de una condición de consulta para estos atributos, tal y como se hizo en la consulta del ejemplo III.2.2.1.

III.2.3. CONCORDANCIA ENTRE CONSULTAS Y MÉTODOS DE BÚSQUEDA

Para que una consulta base `Q` pueda ser ejecutada a través de un determinado método de búsqueda `S`, deben cumplirse dos condiciones:

1) Todas las 4-uplas negativas de `S` deben ser *satisfechas* por las *condiciones de consulta* de `Q`.

Una 4-upla negativa (o restricción) `(A,OP,MUL,VALORES)` *es satisfecha* si la consulta contiene al menos un número `MUL` de condiciones de consulta de la forma `(A OP a1)`, donde `a1 ∈ VALORES`. Por ejemplo, la consulta base del ejemplo III.2.2.1 satisface las restricciones del método de búsqueda del ejemplo III.2.1.2 ya que existe al menos una condición de consulta involucrando el atributo `TITULO` con el operador `CONTIENE`.

2) Todas las condiciones de consulta de Q *son permitidas* por las 4-uplas positivas de S.

Para verificar que las condiciones de Q son permitidas por las 4-uplas positivas de S, aplicamos el siguiente algoritmo:

```
Entradas: CQ contiene las condiciones de consulta de Q.
          CP contiene las 4-uplas positivas de S.

Para cada condición ci = (A OP a) ∈ CQ {
   Si existe una 4-upla positiva pi = (A, OP, MUL, VALORES)∈ CP, tal
que:(MUL >= 1) and (a ∈ VALORES)
   entonces {
      Marcar ci como permitida
      Si (MUL != +) entonces MUL = MUL - 1;
   }
}

Salidas: Si al acabar el proceso, ∀ ci ∈ CQ, ci está marcada como
permitida, entonces la salida es 'verdadero'. En caso contrario, la
salida es 'falso'.
```

La idea básica del algoritmo es sencilla: se itera sobre las condiciones de consulta, buscando alguna 4-upla positiva que permita su ejecución. Es necesario tener en cuenta que cada 4-upla sólo puede ser utilizada tantas veces como permita su multiplicidad (a menos que esta sea +, indicando que puede utilizarse un número arbitrario de veces). Si todas las condiciones son satisfechas, la salida del algoritmo se verifica. En caso contrario, no lo hace.

Así, por ejemplo, la consulta base del ejemplo III.2.2.1 no es permitida por las restricciones del método de búsqueda del ejemplo III.2.1.2 ya que no existe ninguna 4-upla que permita aplicar condiciones de consulta sobre el atributo EDITORIAL.

III.2.3.1. EXTENDIENDO LAS CAPACIDADES DE CONSULTA DE UNA FUENTE MEDIANTE EL USO DE POST-PROCESADOS

III.2.3.1.1. TÉCNICAS DE POST-PROCESADO

A la hora de considerar las capacidades de consulta de una fuente, es también preciso tener en mente que el sistema mediador puede realizar operaciones de post-procesado sobre los resultados obtenidos.

Ejemplo III.2.3.1: Uso de post-procesados

La consulta mostrada en el Ejemplo III.2.2.1 sí puede en realidad ser ejecutada a través del método de búsqueda mostrado en el Ejemplo III.2.1.2, a pesar de que no

todas sus condiciones de consulta sean permitidas por las 4-uplas positivas. El método que se puede emplear para hacer esto es el siguiente:

1) Ejecutar sobre la fuente la consulta (esta sí permitida por el método de búsqueda):

```
select * from R where (titulo contiene 'java') and (precio <= 5000)
```

2) Post-procesar los resultados obtenidos para excluir de los mismos las tuplas que no verifiquen la condición de consulta adicional:

```
(editorial contiene 'Addison-Wesley')
```

□

Existe otra técnica de post-procesado que puede utilizarse para aumentar las capacidades de consulta de una relación, que aprovecha la propiedad de *contención* entre operadores. Diremos que un operador `OP2` *está contenido* en otro operador `OP1`, si `OP1` puede implementar `OP2` mediante post-procesados. Por ejemplo, una condición de consulta que use el operador '=', puede ser implementada utilizando la misma condición de consulta con el operador `CONTIENE`, y aplicando sobre los resultados un filtro que excluya aquellas tuplas obtenidas en el resultado que no satisfagan la condición de igualdad. Por lo tanto diremos que el operador '=' *está contenido* en el operador `CONTIENE`.

Una consideración que es necesario hacer sobre estas técnicas es que sólo pueden llevarse a cabo si los atributos que aparecen en las condiciones de consulta no permitidas por la fuente aparecen en el conjunto de atributos de salida (ya que si no el post-procesado será imposible de realizar). Hay una regla que nos permite extender los atributos de una relación que se encuentran en la salida. En concreto, si un atributo aparece en una 4-upla negativa con el operador '=', entonces puede considerarse que está en la salida, aunque la fuente no lo ofrezca en sus resultados. Para entender esto, considérese el siguiente ejemplo:

Ejemplo III.2.3.2: Considérese una relación `R(A,B,C)` que tiene una 4-upla negativa de la forma `(C,=,MUL,VALORES)`. Esto garantiza que todas las consultas válidas emitidas sobre `R` incluirán una condición de consulta de la forma `(C=c1)` (o, en caso contrario, no se satisfarían las restricciones negativas de `R`). Por lo tanto, podemos asumir que `C` está en el conjunto de salida, tomando el valor `c1` para todas las tuplas.

□

III.2.3.1.2. EXTENSIÓN DE LAS CAPACIDADES DE UNA RELACIÓN MEDIANTE EL USO DE POST-PROCESADOS

En general, cualquier condición de consulta que se aplique sobre un atributo que esté presente en el conjunto de atributos de salida, podrá ser realizada mediante post-procesados.

Esto quiere decir que siempre será posible añadir a las capacidades positivas de un método de búsqueda una 4-upla de la forma `(ATRIBUTO, ANY, +, ANY)` para cada atributo contenido en dicho conjunto. Tal y como ya se ha comentado, una 4-upla como esta hace innecesario que existan otras 4-uplas para el atributo, ya que permite directamente todas las condiciones posibles sobre el mismo. Como se ha comentado también anteriormente, el que un atributo esté en la salida también permite aplicar sobre él cualquier condición booleana, aunque utilice operadores diferentes del *and*.

Por otra parte, cualquier 4-upla negativa `(ATRIBUTO, OP, MUL, VALORES)` que afecte a un atributo que se encuentre en la salida podrá ser satisfecha no sólo por condiciones de la forma `(ATRIBUTO OP a)`, donde `a ∈ VALORES`, sino también por condiciones de la forma `(ATRIBUTO OP2 a)`, donde `a ∈ VALORES` y además `OP2` está contenido en `OP`.

Por la misma razón, una 4-upla positiva `(ATRIBUTO, OP, MUL, VALORES)`, dónde `ATRIBUTO` aparece en la salida, permite una condición de consulta `(ATRIBUTO OP2 a)`, donde `a ∈ VALORES` y además `OP2` está contenido en `OP`.

Ejemplo III.2.3.3: Método de búsqueda extendido con post-procesados para `R`.

```
{
  NEGATIVAS {
     (TITULO, CONTIENE, 1, CUALQUIERA)
  }

  POSITIVAS {
    (TITULO, CUALQUIERA, +, CUALQUIERA)
    (AUTOR, CUALQUIERA, +, CUALQUIERA)
    (FORMATO, CUALQUIERA, +, CUALQUIERA)
    (PRECIO, CUALQUIERA, +, CUALQUIERA)
    (EDITORIAL, CUALQUIERA, +, CUALQUIERA)
  }

 SALIDA {TITULO, AUTOR, EDITORIAL, FORMATO, PRECIO}
}
```

III.2.3.1.3. EFICIENCIA DE CONSULTAS REALIZADAS MEDIANTE POST-PROCESADOS

La técnicas de post-procesado sobre relaciones base (es decir, aquellas que representan directamente a las fuentes y que, por lo tanto, sólo están accesibles a través de la red), sólo deben ser aplicadas cuando sean estrictamente necesarias, ya que pueden ser significativamente menos eficientes que la alternativa en la cual es la fuente la que aplica todas las condiciones de la consulta.

Ejemplo III.2.3.4: Considérese nuevamente la relación `R` definida en los ejemplos III.2.1.1 y III.2.1.2.

Consideremos ahora la consulta:

```
select * from R where (titulo contiene 'java') and (autor contiene
'Naughton')
```

Una posible estrategia para su ejecución sería:

1) Ejecutar sobre la fuente la consulta:

```
select * from R where (titulo contiene 'java')
```

2) Aplicar un post-procesado para quedarse sólo con las tuplas que cumplan la condición adicional:

```
(Autor Contiene 'Naughton')
```

Sin embargo, esta estrategia tiene el grave inconveniente de que la fuente deberá transmitir por la red la respuesta a la primera consulta, que por ejemplo en este caso concreto, puede consistir en varios cientos de libros distribuidos en varias páginas (cada página contendrá típicamente sólo 10-20 resultados). Eso quiere decir que el envoltorio encargado de resolver la consulta sobre la fuente tendrá que realizar tantas peticiones HTTP[HTT97] como páginas de resultados y obtener la respuesta de todas ellas a través de Internet. Lógicamente, el coste temporal de todo este proceso será muy alto.

Sin embargo, permitiendo que sea la fuente la que ejecute la consulta completa, el número de tuplas que la fuente deberá enviar al mediador será mucho menor (para este caso concreto, por ejemplo, probablemente se tratará de una sola página de resultados).
□

Por ello, para el caso de relaciones base, el sistema distinguirá entre las capacidades *nativas* (es decir aquellas que pueden realizarse sin post-procesados) y las *extendidas* (aquellas que pueden ser realizadas sólo mediante post-procesados). Esto será tenido en cuenta posteriormente en los mecanismos de ejecución y optimización.

III.2.3.2. ESTRUCTURAS NO PLANAS

Como se ha comentado, el sistema permite definir atributos de tipo registro y array, que no son tipos de datos planos.

Un tipo de datos plano es aquel cuyos elementos de datos pueden ser únicamente elementos atómicos. Estos son los únicos tipos de datos disponibles en las bases de datos relacionales clásicas.

Otras bases de datos, como las orientadas a objetos, son capaces de trabajar con tipos de datos capaces de representar elementos en árbol. Las modernas bases de datos relacionales también han introducido estos conceptos, aunque se aparten del concepto estricto de "sistema relacional".

Estas estructuras pueden ser contempladas en el sistema añadiendo dos tipos especiales de atributo, cuyos valores no son atómicos:

- Atributos tipo registro. Son aquellos atributos que se subdividen a su vez en campos o atributos. Por ejemplo, podemos tener un atributo '*dirección*' que se subdivida en 3 campos o subatributos llamados '*calle*','*número*' y '*piso*'.
- Atributos tipo array (o multivaluados). Son aquellos atributos que toman como valor un conjunto de elementos de datos y no un dato atómico. Por ejemplo, podemos considerar un atributo '*teléfonos*' cuyo valor sea un conjunto de cadenas de caracteres representando los diferentes números de teléfono de una persona.

Los dos tipos pueden combinarse entre sí, pudiendo así tener, por ejemplo, un atributo de tipo registro y además multivaluado, lo que constituye una *subrelación*.

Los dos siguientes epígrafes tratan, respectivamente, ambos tipos. Se explica como describir y tratar relaciones que presenten dichos tipos de datos.

III.2.3.2.1. TIPO REGISTRO

Los distintos campos de un registro pueden ser accedidos en una consulta con la notación '*nombre_atributo.nombre_campo*'. Por ejemplo: '*direccion.calle*'.

Es importante destacar que dos elementos de tipo registro no pueden ser comparados directamente. Por ejemplo, para determinar si dos elementos de tipo registro son iguales es necesario comparar todos sus campos separadamente.

Dado que dos elementos de tipo registro no pueden ser comparados directamente, un atributo registro no puede ser un atributo de join. En cambio, un campo atómico de un atributo de tipo registro, sí puede serlo.

Por lo tanto, para el proceso de descripción de las capacidades de consulta, los diferentes campos de un registro pueden ser considerados como atributos separados.

III.2.3.2.2. TIPO_ARRAY (O MULTIVALUADO)

Los distintos campos de un registro pueden ser accedidos en una consulta con la notación '*nombre_atributo[índice]*', donde índice es un número entre 0 y (n-1), siendo n el número de elementos en el conjunto. Por ejemplo: '*telefono[0]*' referencia al primer valor del atributo multivaluado '*teléfono*'.

Es importante destacar que, al igual que con los registros, dos elementos de tipo array no pueden ser comparados directamente. Por ejemplo, para determinar si dos elementos de tipo array son iguales es necesario comparar todos sus elementos separadamente.

Dado que dos elementos de tipo array no pueden ser comparados directamente, un atributo array no puede ser un atributo de join.

Un atributo de tipo array puede ser considerado como una relación cuyas tuplas son los diferentes elementos de su conjunto de valores. Por lo tanto, al definir un atributo de tipo array, debemos definir su esquema como si se tratase de una relación (esto es, una *subrelación*). Las columnas de dicha relación pueden accederse como elementos de tipo registro.

Ejemplo III.2.3.5: Consideremos una relación representando a los empleados de una determinada empresa definida como:

```
E = {Nombre: String,
       Telefonos: Array of register {número: String}
     }
```

siendo `Telefonos` un atributo de tipo array, con una columna llamada `numero` cuyos elementos son cadenas de caracteres.
La consulta:

```
Select * from E where telefonos.numero = '981167000'
```

devolverá las tuplas de E representando a aquellos empleados cuya lista de teléfonos incluya el número '`981167000`'.
□

Dado que para definir un tipo array siempre definiremos un tipo registro asociado (los "atributos" de la "sub-relación" representada por el array), para la representación de las capacidades de consulta, los atributos de tipo array pueden descomponerse en los atributos que componen su tipo registro básico y considerar estos como si fuesen atributos adicionales.

Ejemplo III.2.3.6: Supongamos una relación:

```
R={TITULO:String,AUTOR:Array of
register{NOMBRE:String},PRECIO:Money},
```

donde el atributo AUTOR es de tipo array y puede contener varios autores para cada libro. El formulario de búsqueda tiene una caja de búsqueda para el TITULO, que debe rellenarse siempre, y otra opcional para el AUTOR. La búsqueda por AUTOR se verificará para un libro si alguno de sus autores contiene la cadena de búsqueda introducida. El método de búsqueda asociado a esta fuente puede, por lo tanto, representarse así:

```
{
 NEGATIVAS{
   TITULO, CONTIENE, 1, CUALQUIERA)
 }
 POSITIVAS {
  (TITULO, CONTIENE, 1, CUALQUIERA)
  (AUTOR.NOMBRE, CONTIENE,1, CUALQUIERA)
 }
 SALIDA {TITULO,AUTOR,PRECIO}
}
```

□

III.2.4. DEFINIENDO LAS RELACIONES DEL ESQUEMA GLOBAL

Una vez definidas las relaciones base, las relaciones del esquema global (o mediado) serán relaciones derivadas, que estarán especificadas por una vista expresada en álgebra relacional (enfoque Global As View. Ver apartado II.5.1.1). También es posible utilizar relaciones derivadas obtenidas previamente para definir, a su vez, nuevas vistas sobre ellas.

Las operaciones que utilizaremos para definir vistas serán: proyección (incluyendo operaciones de ordenación y agregación), selección (equivalente a la aplicación de una consulta base), unión y "join".

Ejemplo III.2.4.1: Definición de una relación derivada.

Considérense las tres relaciones base siguientes:

```
A ={TITULO:String,AUTOR:String,PRECIO:Money}
B={TITULO:String,AUTOR:String,PRECIO:Money}
C={TITULO:String,AUTOR:String,PUNTUACION:Integer}
```

A y B representan dos tiendas electrónicas de libros en Internet. C representa una fuente que permite a sus usuarios escribir y consultar reseñas de libros. En este caso la fuente se utiliza para consultar la puntuación media que los usuarios han concedido a un determinado libro. Supóngase que deseamos obtener la siguiente relación:

```
R={TITULO:String, AUTOR:String, PRECIO:Money, PUNTUACION:Integer}
```

que contendrá los libros de A y B (sólo una tupla por cada libro), junto con su puntuación media de acuerdo a C, y el precio mínimo para ese libro de los encontrados en ambas tiendas. La definición para R sería:

```
SELECT  TITULO,AUTOR,PUNTUACION, MIN(PRECIO)
FROM (I SELECT TITULO,AUTOR,PRECIO
        FROM A
        UNION
        SELECT TITULO,AUTOR,PRECIO
        FROM B),C
WHERE  I.TITULO=C.TITULO AND I.AUTOR=C.AUTOR
GROUPBY  TITULO,AUTOR,PUNTUACIÓN
□
```

III.2.4.1. CAPACIDADES DE CONSULTA DE RELACIONES DERIVADAS

El principal aspecto involucrado en la definición de relaciones derivadas, es la obtención automática de sus capacidades de consulta a partir de las de las relaciones base de las cuales se componen. Dado que las relaciones del esquema global del mediador (a las que llamamos *relaciones globales*), no son más que el subconjunto de las relaciones derivadas de más alto nivel, entonces el proceso de obtención de las capacidades del sistema mediador se reduce a este proceso.

De esta manera, podremos saber qué consultas pueden realizarse contra el esquema global dadas las capacidades concretas de las fuentes. Esto es de gran importancia ya que, como ya se ha comentado:

1) Permite que los usuarios sepan *a priori* qué consultas pueden ser contestadas por el mediador.

2) Permite que un mediador actúe como fuente para otros mediadores, permitiendo así enfoques incrementales en proyectos grandes de integración de datos.

Además, tal y como se verá más adelante, este proceso se aprovechará también para generar la información que constituirá la base gracias a la cual será posible generar los posibles planes de ejecución de las consultas sobre el esquema global.

El proceso para calcular los métodos de búsqueda se basa en el árbol de la expresión de álgebra relacional que define la vista. Definiremos una serie de reglas para obtener los métodos de búsqueda para un nodo del árbol, partiendo de su tipo de nodo (unión, join, selección y proyección), y de los métodos de búsqueda de sus nodos hijos.

Una vez definidas estas reglas, obtener las capacidades de consulta de una relación definida según una determinada expresión de álgebra relacional, se reducirá a construir el árbol de la citada expresión algebraica e ir recorriéndolo desde abajo hacia arriba,

"propagando" en él las capacidades de consulta, mediante la aplicación en cada nodo del proceso asociado a la operación de álgebra relacional que ese nodo represente. Los métodos de búsqueda obtenidos para el nodo raíz serán los de la relación.

El modo de aplicación de las reglas que se proporcionarán variará en función de si el nodo sobre el que van aplicarse representa una operación n-aria (union, join) o unaria (selección, proyección).

En el caso de las operaciones unarias, se generará un nuevo método de búsqueda para el nodo padre por cada método de búsqueda del nodo hijo.

En el caso de las operaciones n-arias, bastará considerar el caso en el que el número de nodos hijos es dos. Si en lugar de dos, se desease realizar la operación para un número de fuentes arbitrario *n*, basta realizar *(n-1)* veces el proceso con las dos fuentes.

Ejemplo III.2.4.1: Si tenemos 4 fuentes `F1`, `F2`, `F3` y `F4`, y queremos hacer su unión, entonces podemos computar:

```
1) R1= (F1 ∪ F2)
2) R2= (F3 ∪ R1)
3) R3= (F4 ∪ R2)
```

y `R3` es el resultado final.
□

Para el caso de dos nodos hijos, se generará un nuevo método de búsqueda por cada par en el producto cartesiano de los métodos de búsqueda de los dos nodos hijos (a los que llamaremos *métodos de búsqueda base*). Por lo tanto, en los apartados posteriores daremos únicamente las reglas para obtener el nuevo método de búsqueda para cada par de métodos base.

III.2.4.1.1. VISTAS CONSTRUIDAS MEDIANTE LA 'UNIÓN'

Una primera consideración es que para realizar la unión de dos relaciones, precisamos que ambas tengan el mismo esquema (si no fuese así e interesase realizar la unión igualmente, bastaría con añadir a cada relación los atributos que le faltan, suponiéndoles el valor `null` para todas las tuplas)

III.2.4.1.1.1. Conjunto de atributos de salida

Un atributo estará en la salida del nuevo método de búsqueda si está en la salida de los dos métodos base.

Si un atributo está en la salida de una de las dos fuentes base y no está en la salida de la otra, el atributo no estará en la salida, a efectos de calcular las capacidades de

consulta. De lo contrario, podría ocurrir que se devolviesen resultados inexactos en las consultas al aplicar post-procesados sobre el atributo.

Sin embargo, a efectos, de devolución de los resultados, el atributo sí estará en la salida tomando el valor NULL para aquellas tuplas obtenidas de la fuente que no devuelve el atributo.

III.2.4.1.1.2. Capacidades negativas

Para entender adecuadamente los siguientes pasos, es conveniente tener en mente el proceso que seguirá el mediador para ejecutar una consulta realizada sobre el nodo unión: sencillamente ejecutará la consulta recibida sobre ambos nodos hijos y realizará la unión de ambos sub-resultados.

Una primera consecuencia de esto es que las capacidades negativas de un método base deben estar permitidas por las capacidades positivas del otro método base, y viceversa. En otro caso, el mediador no podría ejecutar ninguna consulta sobre el nodo unión ya que deberían cumplirse a la vez los dos siguientes requisitos, que son contradictorios entre sí:

1) Ambos métodos de búsqueda base deben ejecutar las mismas consultas
2) Uno de los métodos base exige que en las consultas recibidas estén presentes condiciones de consulta con unas características que el otro método de base exige explícitamente que no estén.

Suponiendo que la condición anterior es satisfecha, entonces las capacidades negativas del nuevo método de búsqueda se obtienen simplemente incluyendo las de los dos métodos base (ya que las consultas recibidas por el nodo unión, al tener que ser ejecutadas sobre ambos métodos base, tendrán que satisfacer las restricciones de ambos); y excluyendo las 4-uplas redundantes (ver sección III.2.1.1.1).

III.2.4.1.1.3. Capacidades Positivas

Tal y como ya se ha comentado, gracias a los post-procesados, para todos los atributos que estén en el conjunto de salida, se puede añadir una 4-upla `(ATRIBUTO, CUALQUIERA, +, CUALQUIERA)`.

Para los atributos no presentes en la salida, se calcula el producto cartesiano de las 4-uplas del atributo en ambos métodos base. Para cada par `(X,Y)` en dicho producto cartesiano:

```
X=(A,OP,MUL1,VALORES1)
Y=(A,OP,MUL2,VALORES2)
(nótese que para que se genere una nueva 4-upla, ambas 4-uplas base
deben tener el mismo operador)
```

crearemos una nueva 4-upla de la forma:

(A,OP,MIN(MUL1,MUL2),VALORES1∩VALORES2)

III.2.4.1.2. VISTAS DE TIPO JOIN

A diferencia de en el caso de la unión, el mediador puede utilizar varias estrategias de ejecución posibles. Como consecuencia de esto, las capacidades de consulta calculadas para los nodos join variarán dependiendo del método de ejecución utilizado.

Por ello, en el momento de generación de la vista, deben calcularse los métodos de búsqueda para todas las estrategias de ejecución posibles. Más tarde, en tiempo de ejecución, el mediador escogerá la estrategia a utilizar en función de su coste computacional estimado y de las capacidades de consulta que permite.

Consideraremos dos estrategias de ejecución de joins: *join convencional* y *join anidado*.

Una consulta sobre un nodo de join utilizando la estrategia convencional se realiza enviando una sub-consulta a cada nodo hijo y, posteriormente, realizando el join de ambos sub-resultados. La sub-consulta de cada nodo incluirá las condiciones de consulta referentes a los atributos de su relación (las referentes a atributos de join irán, por tanto, en ambas sub-consultas ya que, por definición, son comunes a ambas sub-relaciones).

En cambio, en la estrategia anidada, se ejecuta en primer lugar una sub-consulta sobre el primer nodo hijo y, entonces, para cada valor único devuelto en el sub-resultado para los atributos de join, se envía una sub-consulta sobre el segundo nodo para obtener las tuplas de la segunda relación que toman dicho valor para los atributos de join.

Ejemplo III.2.4.2: Join anidado

Consideremos dos relaciones R1(A,B,C) y R2(A,D,E). Supongamos también una relación R3(A,B,C,D,E) definida por $R1 \infty_A R2$. Si se recibe la consulta:

```
Select * from R3 where (A op1 value1) and (B op2 value2) and (D op3
value3)
```

entonces la estrategia de join anidado para ejecutar esta consulta se aplicaría como sigue:

1) Se envía sobre R1 la siguiente consulta Q1 :

```
Q1 = Select * from R1 where (A op1 valor1) and (B op2 valor2)
```

2) Entonces, para cada valor único ai para el atributo A, de los obtenidos en Q1, se envía la siguiente consulta sobre R2:

$Q2_i$= Select * from R2 where (A = a_i) and (D op3 valor3)

3) Se calcula la unión de todos los resultados parciales obtenidos de R2:

$Q2 = Q2_1 \cup Q2_2 \cup \ldots \cup Q2_n$

4) Finalmente, se calcula $Q1 \infty_A Q2$ y se obtiene el resultado final.
□

III.2.4.1.2.1. Join convencional

III.2.4.1.2.1.1. Conjunto de atributos de salida

Para que sea posible ejecutar el join según esta estrategia, es necesario que los atributos de join estén en la salida de los dos métodos base. En otro caso, no será posible crear el nuevo método de búsqueda.

Si esta condición se cumple, el conjunto de salida para el nuevo método se obtiene realizando la unión de los conjuntos de salida de los métodos base.

III.2.4.1.2.1.2. Capacidades negativas

Debido a que las condiciones sobre los atributos de join irán en las sub-consultas enviadas a ambos nodos hijos, las nuevas 4-uplas para los atributos de join se obtienen de la misma manera que se describió para todos los atributos del caso de la unión.

Para el resto de atributos, basta con copiar las 4-uplas de los métodos base.

III.2.4.1.2.1.3. Capacidades positivas

Como es habitual, se incluye una 4-upla (A,CUALQUIERA,+,CUALQUIERA) para todos los atributos presentes en el conjunto de salida.

En cuanto a los atributos no presentes en el conjunto de salida, ya hemos visto que no puede ocurrir que sean atributos de join (o si no, no podría crearse el nuevo método de búsqueda). Las nuevas 4-uplas de los atributos que no sean de join y no estén en el conjunto de salida, se obtienen copiando las de los métodos base.

III.2.4.1.2.2. Join anidado

Una peculiaridad de esta estrategia de ejecución es que, tal y como hemos visto en el ejemplo III.2.4.2, el orden en el cual se consideran los nodos hijos es relevante.
Por lo tanto, precisamos considerar los dos casos en los que cada uno de los nodos hijos es escogido como primera relación.

Para cada uno de estos dos casos, se generará un nuevo método de búsqueda para cada par en el producto cartesiano de los conjuntos de métodos de búsqueda base. Para cada método generado, el sistema almacenará el orden de las relaciones que fue considerado para su generación.

III.2.4.1.2.2.1. Conjunto de atributos de salida

Para que un join pueda ejecutarse siguiendo esta estrategia, es necesario que los atributos de join estén en el conjunto de salida del método de búsqueda de la primera relación. En otro caso, no podría crearse el nuevo método de búsqueda. Nótese que no es requerido que los atributos de join estén en la salida de la segunda relación. La razón es que, tal y como se vio en el paso 2 del ejemplo III.2.4.2, el valor de los atributos de join para las tuplas obtenidas de la segunda relación es siempre conocido (ya que va implícito en las sub-consultas a través de las que dichas tuplas son obtenidas).

Si esta condición se cumple, el conjunto de atributos de salida para el nuevo método de búsqueda se obtiene incluyendo los atributos de join y los atributos que no sean de join y que estén en la salida del método base de la relación de la cual provienen. Nótese que aunque los atributos de join no estén presentes en la salida de la segunda fuente, se incluyen igualmente debido a la misma razón ya comentada anteriormente.

III.2.4.1.2.2.2. Capacidades negativas

La diferencia con la estrategia de ejecución convencional es que, para los atributos de join, las 4-uplas de la segunda relación que tengan el operador '=', no deben incluirse en el nuevo método de búsqueda. La razón es que en la estrategia anidada, la segunda relación es consultada siempre utilizando el operador '=' con los valores obtenidos de la primera relación (recuérdese el paso 2 del ejemplo III.2.4.2) y, por lo tanto, siempre es posible satisfacer la restricción representada por esa 4-upla (nótese que, incluso aunque la 4-upla tuviese una multiplicidad mayor que 1, será posible satisfacerla igualmente, ya que al ser el operador involucrado el '=', podría usarse el mismo valor para satisfacer todas las restricciones sin afectar al resultado de la consulta: aplicar una condición de consulta `(A = a)` es equivalente a aplicar `(A = a) and (A = a)and ... and (A = a)`).

III.2.4.1.2.2.3. Capacidades positivas

Una primera consideración es que esta estrategia de join sólo puede ser realizada si la segunda relación admite ser consultada con el operador '=' por todos los atributos de join (recuérdese nuevamente el paso 2 del ejemplo III.2.4.2).

Si la condición anterior se cumple, se incluye una 4-upla `(A,CUALQUIERA,+,CUALQUIERA)` para todos los atributos presentes en el conjunto de salida.

En cuanto a los atributos no presentes en el conjunto de salida, ya hemos visto que no puede ocurrir que sean atributos de join. Las nuevas 4-uplas de los atributos que no sean de join y no estén en el conjunto de salida, se obtienen copiando las de los métodos base.

III.2.4.1.3. NODOS PROYECCIÓN

Como es sabido, una operación de proyección sobre una relación consiste en eliminar de sus tuplas parte de sus atributos (llamados *atributos ocultos*).

III.2.4.1.3.1. Conjunto de atributos de salida

Los atributos del conjunto de salida en el nuevo método de búsqueda serán los mismos que los del método de búsqueda base, exceptuando a los atributos ocultos.

III.2.4.1.3.2. Capacidades negativas

Si en el método de búsqueda base existe una 4-upla negativa para algún atributo oculto, el nuevo método de búsqueda no podrá ser creado.

En otro caso, las 4-uplas del nuevo método se obtienen simplemente copiando las del método base.

III.2.4.1.3.3. Capacidades positivas

Se incluye una 4-upla `(A,CUALQUIERA,+,CUALQUIERA)` para todos los atributos presentes en el conjunto de salida.

Las 4-uplas de los atributos no presentes en la salida se copian desde el método de búsqueda base.

Las 4-uplas de los atributos ocultos se eliminan.

III.2.4.1.4. NODOS SELECCIÓN

Como es sabido, un nodo de selección filtra las tuplas de la relación hija para quedarse solamente con aquellas que verifican una cierta *operación de selección* boleana.

En nuestro sistema, una operación de selección tiene el mismo formato que una consulta base emitida sobre la relación hija.

III.2.4.1.4.1. Conjunto de atributos de salida

Los atributos del conjunto de salida del nuevo método de búsqueda incluirán a todos los del método base. Además, se incluirán aquellos atributos no presentes en el conjunto de salida del método base que aparezcan en una condición de la operación de selección con el operador '='.

Ejemplo III.2.4.3: Considérese una relación `R(A,B,C)` donde el atributo `C` no está presente en los resultados. Supóngase ahora que se define una vista de selección `R'` sobre `R`, con la operación de selección `(C=c1)`. En la práctica, podemos considerar que `C` se encuentra en la salida de `R'`, ya que su valor será conocido para todas las tuplas (forzosamente será `c1`).

□

III.2.4.1.4.2. Capacidades negativas

Las 4-uplas negativas para el nuevo método de búsqueda se obtienen copiando las del método base y excluyendo aquellas que son satisfechas por las condiciones de la operación de selección (si una restricción es satisfecha ya por la operación de selección, no es necesario ya que sea satisfecha por la consulta y, por lo tanto, puede suprimirse).

III.2.4.1.4.3. Capacidades positivas

En primer lugar hay que decir que, para que el nuevo método de búsqueda pueda ser creado, es necesario que las capacidades positivas del método base permitan la ejecución de la operación de selección. Para comprobar esto, puede usarse directamente el algoritmo presentado en la sección III.2.3 para comprobar si las condiciones de una consulta dada (la operación de selección, en este caso) son permitidas por las 4-uplas positivas de un determinado método de búsqueda.

Para los atributos no presentes en la salida, las nuevas 4-uplas serán las mismas obtenidas después de aplicar el algoritmo mencionado. Nótese que la multiplicidad de las 4-uplas puede verse disminuida como resultado de este proceso, ya que las condiciones de la operación de selección pueden "gastar" parte de las capacidades de consulta obtenidas (esto es,

Para los atributos presentes en la salida, basta incluir una 4-upla `(A,CUALQUIERA,+,CUALQUIERA)`.

III.2.4.1.5. MINIMIZACIÓN DEL CONJUNTO DE CAPACIDADES DE CONSULTA

Con un número elevado de fuentes y de métodos de búsqueda, rápidamente se ve que los métodos descritos pueden acabar ofreciendo como resultado una cantidad bastante alta de métodos de búsqueda, posiblemente redundantes.

Un objetivo a cumplir es describir las capacidades de consulta del sistema mediador con el menor número posible de métodos de búsqueda. Este apartado describe una técnica sencilla que puede utilizarse para dicho propósito.

Una nota importante sobre el proceso de minimización es que el mediador la aplicará sólo a efectos de ofrecer una representación más compacta de capacidades. A

efectos de ejecución, el sistema mediador no eliminará los métodos de búsqueda redundantes, ya que cada método de búsqueda *redundante* es en realidad una alternativa diferente para ejecutar una consulta determinada. Es decir, cada método constituye un posible plan de ejecución. Como quiera que el plan de ejecución óptimo para una consulta puede ser diferente del óptimo para otra, es conveniente que el sistema pueda considerarlos todos.

Diremos que un método de búsqueda B *es redundante* con respecto a un método de búsqueda A si todas las consultas que pueden a realizarse a través de B pueden también realizarse a través de A. En esas circunstancias, B puede ser suprimido de la lista de métodos de búsqueda de la relación.

Esto ocurrirá en el caso de que para cada atributo de la relación, las 4-uplas negativas de B sean al menos tan restrictivas como las correspondientes 4-uplas en A, y además las 4-uplas positivas para el atributo en A permitan todas las consultas permitidas por las de B. En III.2.1.1.1 fueron introducidas unas reglas que permiten saber si una 4-upla positiva o negativa es redundante con respecto a un conjunto de 4-uplas, que pueden utilizarse aquí sin cambios para detectar métodos de búsqueda redundantes.

Es importante destacar que estas técnicas no garantizan que se vayan a encontrar todas las redundancias: simplemente permiten detectar la mayoría de los casos más comunes. Un algoritmo que garantizase la detección de todas las redundancias (es decir, que nos permitiese cambiar los "si" de los párrafos precedentes por " si y solo si"), forma parte de nuestro trabajo futuro.

III.2.4.1.6. EJEMPLO

Supongamos que queremos definir en el sistema mediador una base de datos que nos permita comparar precios de libros en Internet y además acceder a reseñas de los mismos.

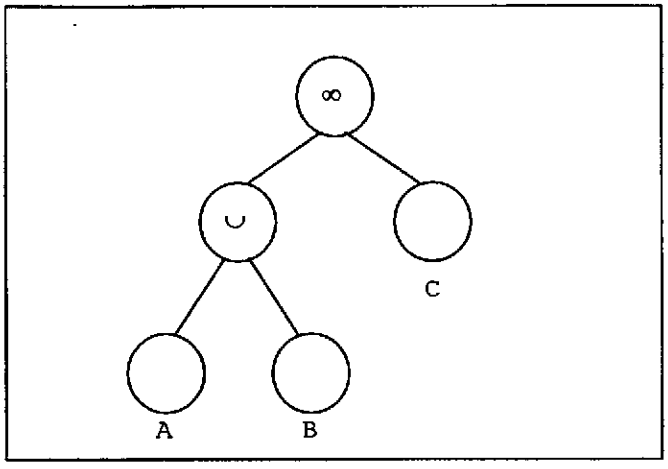
Consideraremos 3 fuentes para este ejemplo: 2 tiendas virtuales de libros a las que llamaremos A y B y la web de un periódico on-line que proporciona acceso web a una base de datos de puntuaciones de libros según la opinión de sus críticos, que denominaremos C. Los esquemas de dichas fuentes son los siguientes:

```
A={TITULO:String,AUTOR:String, PRECIO:Money}
B={TITULO: String, AUTOR:String, PRECIO:Money}
C={TITULO:String, AUTOR:String, PUNTUACION:Integer}
```

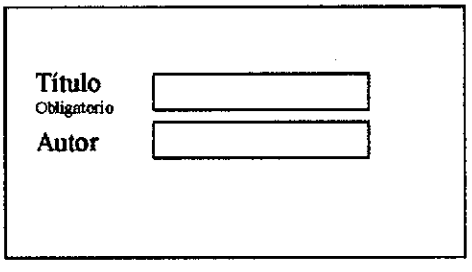
El esquema global G deseado es el definido por la vista expresada como:

$G = (A \cup B) \infty_{TITULO,AUTOR} C$ = {TITULO: String, AUTOR: String, PRECIO: Money, PUNTUACIÓN:Integer}

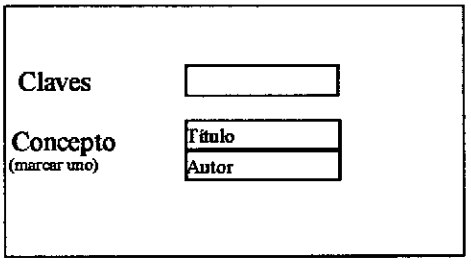
El árbol que representa esta expresión sería el siguiente:



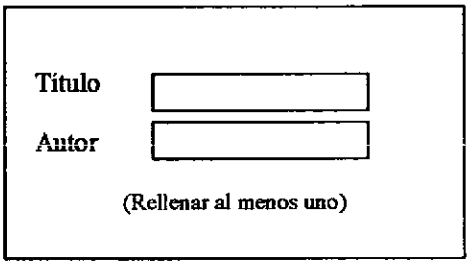
A continuación se muestran los formularios de búsqueda de las tres fuentes:



Formulario de A



Formulario de B



Formulario de R

La tienda A admite consultas que obligatoriamente incluyan el título de un libro y opcionalmente pueden incluir el autor.

La tienda B admite consultas o bien por título o bien por autor según se escoja el valor del campo 'Concepto' del formulario.

La fuente R admite consultas o bien por título o bien por autor (con operador de igualdad en este caso). Al menos uno de los dos campos debe rellenarse. El atributo AUTOR no aparece en los resultados.

Teniendo esto en cuenta, se obtienen las capacidades de consulta para cada fuente, que se muestran a continuación.

```
Métodos de búsqueda de A

  {
   NEGATIVAS {
      (TITULO, CONTIENE, 1, CUALQUIERA)
   }
   POSITIVAS{
      (TITULO, CONTIENE, 1, CUALQUIERA)
      (AUTOR, CONTIENE, 1, CUALQUIERA)
   }
   SALIDA {TITULO, AUTOR, PRECIO}
  }

Métodos de búsqueda de B

  {
   NEGATIVAS {
      (TITULO, CONTIENE, 1, CUALQUIERA)
   }
   POSITIVAS{
      (TITULO, CONTIENE, 1, CUALQUIERA)
   }
   SALIDA {TITULO, AUTOR, PRECIO}
  }

  {
   NEGATIVAS {
      (AUTOR, CONTIENE, 1, CUALQUIERA)
   }
   POSITIVAS{
      (AUTOR, CONTIENE, 1, CUALQUIERA)
   }
   SALIDA {TITULO, AUTOR, PRECIO}
  }

Métodos de búsqueda de R

{
   NEGATIVAS {
```

```
      (TITULO, =, 1, CUALQUIERA)
   }
   POSITIVAS{
      (TITULO, =, 1, CUALQUIERA)
   }
   SALIDA {TITULO, PUNTUACION}
  }

{
   NEGATIVAS {
      (AUTOR, =, 1, CUALQUIERA)
   }
   POSITIVAS{
      (AUTOR, =, 1, CUALQUIERA)
   }
   SALIDA {TITULO, PUNTUACION}
  }
```

Nótese como en la fuente R no puede decirse que ninguno de los dos atributos (TITULO y AUTOR) sea obligatorio en las consultas. Sin embargo, al menos uno de ellos sí debe rellenarse. Para representar esto, deben crearse dos métodos de búsqueda, uno en el que el atributo TITULO aparece en las capacidades negativas y AUTOR sólo aparece en las positivas, y otro en el que se invierten los términos, quedando así esta circunstancia correctamente representada.

Es también importante recordar que los post-procesados pueden extender las capacidades de consulta de estas relaciones de la manera que se comentó en al apartado III.2.3.1. En particular, todos los atributos que se encuentren en la salida pueden ser consultados mediante cualquier operador y valor cuantas veces se desee, mediante el uso de post-procesados.

Comenzamos calculando las capacidades de consulta resultantes de realizar la unión $I=A \cup B$.

Combinando el método de búsqueda de A con el 1° de B, aplicando las reglas que hemos visto, obtenemos:

```
{
 NEGATIVAS {
  (TITULO, CONTIENE, 1, CUALQUIERA)
 }
 POSITIVAS {
  (TITULO, CUALQUIERA, +, CUALQUIERA)
  (AUTOR, CUALQUIERA, +, CUALQUIERA)
  (PRECIO, CUALQUIERA, +, CUALQUIERA)
 }
 SALIDA {TITULO, AUTOR, PRECIO}
}
```

Combinando el método de búsqueda de A con el 2° de B obtenemos:

```
{
 NEGATIVAS {
  (TITULO, CONTIENE, 1, CUALQUIERA)
  (AUTOR, CONTIENE, 1, CUALQUIERA)
 }
 POSITIVAS {
  (TITULO, CUALQUIERA, +, CUALQUIERA)
  (AUTOR, CUALQUIERA, +, CUALQUIERA)
  (PRECIO, CUALQUIERA, +, CUALQUIERA)
 }
 SALIDA {TITULO, AUTOR, PRECIO}
}
```

Aplicando las reglas de minimización ya vistas, el segundo método de búsqueda obtenido es redundante con respecto al primero, con lo cual éste último puede suprimirse (aunque como ya se ha comentado, esta supresión del método de búsqueda redundante se realiza sólo a efectos de obtener una representación más compacta, pero será considerado en la práctica como un posible plan de ejecución válido para las consultas efectuadas sobre I).

Una vez obtenidas las capacidades de consulta para $I=A \cup B$, pueden obtenerse ya las de $G=I \infty R$. Por brevedad, consideraremos sólo el método de búsqueda de I no redundante (aunque ya hemos dicho que el mediador proseguiría el proceso con todos).

Para ello, empezamos considerando la estrategia de join convencional. En este caso, esta estrategia no es aplicable ya que el atributo de join AUTOR no se encuentra en el conjunto de atributos de salida del segundo método de búsqueda. Por lo tanto, no puede derivarse un nuevo método de búsqueda con esta estrategia.

Considérese ahora la estrategia de join anidado. Para que sea aplicable deben cumplirse dos requisitos:

- Que los atributos de join estén en la salida de la primera relación.
- Que los atributos de join sean consultables en la segunda relación con el operador '='.

El requisito 1) sólo se cumple si se escoge I como primera relación. El requisito 2) sólo es cumplido por el segundo método de búsqueda de R (nótese que al estar TITULO en la salida de dicho método, es consultable mediante la aplicación de las técnicas de post-procesado).

Por lo tanto, podremos obtener un único método de búsqueda para G (considerando I como primera fuente y combinando el método de búsqueda I con el segundo de R). El método es:

```
{
 NEGATIVAS {
```

```
  (TITULO, CONTIENE, 1, CUALQUIERA)
 }
 POSITIVAS {
  (TITULO, CUALQUIERA, +, CUALQUIERA)
  (AUTOR, CUALQUIERA, +, CUALQUIERA)
  (PRECIO, CUALQUIERA, +, CUALQUIERA)
  (PUNTUACION, CUALQUIERA, +, CUALQUIERA)
 }
 SALIDA {TITULO, AUTOR, PRECIO, PUNTUACION}
}
```

□

III.2.5. EJECUCIÓN DE CONSULTAS

III.2.5.1.1. VISIÓN GENERAL DEL PROCESO DE EJECUCIÖN

Una vez terminada la creación de las relaciones del esquema global, el mediador está listo para aceptar consultas sobre ellas.

Cuando se recibe una consulta, el intérprete de consultas la traduce al formato interno del mediador que, como hemos visto, es básicamente álgebra relacional.

Una vez hecho esto, la consulta puede considerarse como la definición de una vista sobre las relaciones del esquema global. Por lo tanto, el sistema construye dicha vista realizando el mismo proceso que ya se ha visto para definir las relaciones globales, propagación de capacidades de consulta incluida (sólo que en este caso, la propagación se realiza desde las relaciones del esquema global hasta la raíz del árbol de la vista).

Si el resultado de la propagación de las capacidades de consulta es nulo (i.e. no se obtiene ningún método de búsqueda como resultado), entonces eso quiere decir que la consulta no es válida. En caso contrario, se obtendrán un conjunto de métodos de búsqueda (sin minimizar), que son los posibles planes de ejecución para la consulta. Si alguno de los métodos de búsqueda tiene alguna 4-upla en su lista de capacidades negativas, entonces ese método de búsqueda no podrá utilizarse ya que eso quiere decir que las condiciones incluidas en la consulta no llegan para satisfacer todas las restricciones impuestas por las fuentes. Si todos los métodos de búsqueda tuviesen alguna 4-upla en sus capacidades negativas, entonces la consulta no podría contestarse.

Cada método de búsqueda tendrá asociado un árbol algebraico que indica las operaciones a realizar para obtener sus tuplas. Es importante destacar que en la generación del árbol se aplica el conocido principio de "empujar hacia abajo las selecciones y proyecciones" dentro del árbol, de uso común en los sistemas relacionales con el fin de optimizar el proceso de obtención de las tuplas. En nuestro caso, es necesario tener en cuenta las limitaciones en las capacidades de consulta de los nodos, de manera que es posible que la propagación hacia abajo tenga que detenerse, debido a este motivo, en algún nodo anterior a las hojas. Este proceso es especialmente importante en sistemas mediadores, ya que consigue que las fuentes (que son las hojas

del árbol) realicen localmente el mayor trabajo posible, disminuyendo así el tránsito de datos con el mediador. Las proyecciones y selecciones que no hayan podido ser "empujadas" hasta las fuentes tendrán que ser realizadas por el mediador post-procesando los resultados devueltos por las mismas.

En este punto, el sistema dispone ya de un árbol completo para cada posible método de búsqueda. Las hojas de cada árbol son las relaciones base. Por lo tanto, en este punto, se ha completado ya la reformulación de la consulta.

Entonces, el mediador considera todos los posibles métodos de búsqueda generados en el proceso de propagación de capacidades. Cada método de búsqueda es, de hecho, un posible plan de ejecución para la consulta. Tal y como ya se ha visto, los planes diferirán normalmente entre sí por aspectos como la estrategia utilizada para ejecutar los joins (join convencional o join anidado) o los métodos de búsqueda concretos a utilizar sobre las fuentes. Es responsabilidad del módulo *optimizador* obtener el coste de cada uno de los planes, de manera que pueda escogerse el mejor.

Una vez escogido el plan más óptimo, el *motor de ejecución* se encarga de su puesta en práctica. Para ello, a grandes rasgos, recorre el árbol del plan desde la raíz hasta llegar a las hojas del mismo. Tal y como se ha construido el árbol, cada hoja puede verse como una representación de una subconsulta que involucra solamente a una fuente. Cada una de esas subconsultas es enviada al envoltorio de la fuente y ejecutada por el mismo. El acceso a las fuentes se realiza en paralelo.

Entonces, el motor de ejecución combina los resultados devueltos por las fuentes de acuerdo a lo especificado por el árbol del plan, obteniendo así la respuesta final a la consulta. El sistema funciona de manera *asíncrona*. Esto quiere decir que a medida que van estando disponibles resultados de las fuentes, el sistema empieza a intentar procesarlos aunque las fuente aún no hayan emitido una respuesta completa. Esto puede acelerar mucho los tiempos de obtención de las primeras tuplas del resultado final.

Otro aspecto importante es que, opcionalmente, todos o algunos de los envoltorios pueden haber sido configurados para mantener una cache de los datos de las fuentes. En ese caso, el envoltorio comprobará si la subconsulta recibida puede ser contestada con los datos contenidos en la cache. Si es así, obtiene la respuesta directamente de la misma en lugar de consultar a la fuente. De esta manera, y de acuerdo a la terminología habitual en mediadores, el sistema sigue una aproximación básicamente *virtual* (esto es, los datos permanecen en las fuentes y son accedidos dinámicamente por el mediador en cada consulta), pero proporciona también (a través de la cache) funcionalidades para trabajar en modo *warehousing* (esto es, con copias locales al mediador de los datos de las fuentes).

Los siguientes apartados tocan con más detalle los aspectos aquí apuntados. Así, el apartado III.2.5.2 trata en mayor detalle el lenguaje de consultas utilizado. Los apartados III.2.5.3 y III.2.5.4 se ocupan de aspectos relacionados con la optimización.

El apartado III.2.5.5 muestra el algoritmo para la obtención de las tuplas de respuesta a una consulta, dado el árbol del método de búsqueda escogido por el optimizador. Por último, III.2.5.6 se ocupa del sistema de cache.

III.2.5.2. LENGUAJE DE CONSULTAS

Ya se ha dicho que la forma de realizar consultas sobre el esquema mediado tiene una relación directa con la manera de definir vistas. En cierto modo, realizar una consulta determinada no es más que construir una vista y viceversa.

Por lo tanto, dada una consulta sobre el esquema global (es decir, sobre una vista o conjunto de vistas de un mediador), el método para darle respuesta es el mismo que el utilizado para definir y ejecutar vistas. Por lo tanto, se admiten en las consultas los operadores relacionales de proyección, selección, unión y *join*, así como operadores de agregación y ordenación.

De cara al programador de aplicaciones que utilice la Base de Datos, se ha optado por una sintaxis similar a SQL. La sintaxis SQL facilita la formación al permitir usar lenguajes de consulta similares a los habituales en Bases de Datos convencionales.

A continuación se presenta cómo deben de interpretarse las consultas formuladas en SQL. Sean dos relaciones `T1 (A, B)` y `T2 (A, C)`. Sobre ellas se formula la siguiente consulta:

```
SELECT A, B, C
FROM T1, T2
WHERE T1.A=T2.A
          AND A='clave_A'
AND C < 'valor_C';
```

Cuando esta consulta se presenta al mediador, el primer paso consiste en traducirla a una representación interna en álgebra relacional. Los campos presentes en el SELECT de la consulta definen el esquema de la vista resultante; en el FROM se indican las vistas previamente definidas en el catálogo que constituyen la relación sobre la que buscar. Cuando en el FROM de una sentencia SQL aparece más de una relación o vista, se trata de una operación de *join* y entre las condiciones del WHERE deben encontrarse las condiciones de *join*. Las restantes condiciones establecen los parámetros de la selección aplicada sobre el *join*. La expresión en álgebra relacional asociada a la consulta SQL anterior sería:

$$\Pi_{A,B,C}\,\sigma_{A='clave\_A' \wedge C<'valor\_C'}(T_1 \infty_{T_1.A=T_2.A} T_2)$$

En la consulta SQL pueden incluirse también las operaciones de agregación y ordenación utilizando la sintaxis habitual.

Además, la consulta tendrá un contexto asociado, en el que permite indicar, entre otros parámetros, si desea utilizar la *cache* del sistema y en qué nivel (hoja, consulta, etc.), manejadores de resultados (filtros, manejo de heterogeneidades, etc.) o el formato con el que se desea el resultado de la consulta (por ejemplo, de acuerdo al soporte de internacionalización del sistema mediador, pueden escogerse la zona horaria para los atributos de tipo fecha, la divisa para los atributos de tipo dinero o el lenguaje para los atributos de tipo texto).

III.2.5.3. OPTIMIZACIÓN DE CONSULTAS EN ÁLGEBRA RELACIONAL

La principal técnica de optimización de consultas en álgebra relacional es la que mueve las proyecciones y selecciones lo más abajo posible dentro del árbol de una consulta, minimizando así el número de tuplas (selección) y atributos (proyección) a tratar en cada caso. Este apartado ilustra este punto con mayor detalle.

En primer lugar, considérese el siguiente ejemplo:

Ejemplo III.2.5.1: Dada la consulta en álgebra relacional:

$\Pi_{A,B}\,(\sigma_{(A\ \text{Contiene 'a1'})\,(B = b1)}\,((R1 \cup R2))$,

puede optimizarse obteniendo la siguiente expresión:

$(\Pi_{A,B}\,(\sigma_{(A\ \text{Contiene 'a1'})\,(B = b1)}\,(R1))) \cup (\Pi_{A,B}\,(\sigma_{(A\ \text{Contiene 'a1'})\,(B = b1)}\,(R2)))$.

□

En nuestro caso, esta técnica es aplicable con la salvedad de que las proyecciones y selecciones sólo podrán ser desplazadas hacia abajo en el árbol de la consulta si esto es permitido por las capacidades de consulta de los nodos del árbol. Por ejemplo, en el caso anterior, las selecciones sólo podrán pasarse hacia abajo en el árbol si los métodos de búsqueda de `R1` y `R2` permiten realizar directamente sobre ellas la condición: `(A Contiene a1) and (B = b1)`.

En este punto debe recordarse que, tal y cómo se explicó en el apartado III.2.3.1.3, en las relaciones base, se guardan separadamente sus *capacidades nativas* y sus *capacidades extendidas*, de manera que es posible tener esto en cuenta en el momento de la ejecución. Para realizar esto, se sigue el sencillo mecanismo de considerar una vista proyección sin atributos ocultos sobre cada relación base. Al no haber ningún atributo oculto, la vista no afecta al esquema de la relación base. Sin embargo, y de manera coherente con las reglas de propagación de capacidades que hemos visto, la relación base tendrá los métodos de búsqueda correspondientes a sus *capacidades nativas* y la vista proyección tendrá los de las *capacidades extendidas*. De esta manera, la técnica de optimización comentada en este apartado garantiza ya que las capacidades nativas de las fuentes serán utilizadas siempre que sea posible.

La aplicación de esta técnica, al menos en lo que respecta a las selecciones, es muy importante en el peculiar contexto de ejecución que suele estar asociado a los sistemas basados en mediadores, tal y como se mostró en el apartado III.2.3.1.3.

III.2.5.4. CALCULANDO COSTES PARA LOS MÉTODOS DE BÚSQUEDA: OPTIMIZADOR

Como ya se ha comentado, es responsabilidad del módulo optimizador calcular el coste asociado a cada método de búsqueda para cada consulta, de manera que pueda escogerse el plan más óptimo.

Si bien el sistema de optimización del sistema mediador no forma parte del trabajo presentado en esta tesis doctoral, en este apartado se exponen las líneas básicas de su funcionamiento para proporcionar una visión más completa.

La base de la información para cálculo de costes son estadísticas recolectadas por los envoltorios sobre el comportamiento de las fuentes con consultas previas. En función de ello, cada método de búsqueda en una fuente va siendo valorado cuantitativamente para cada tipo de consulta.

Cuando se recibe una consulta y se obtienen todos los posibles planes de ejecución (que se corresponden con todos los métodos de búsqueda obtenidos, sin minimizar, en el proceso de obtención de capacidades de consulta), se comienza asignando un coste a los nodos hoja de cada plan (cada hoja representa un método de búsqueda sobre una fuente), en función de las estadísticas recolectadas. Una vez que los métodos de búsqueda de las relaciones base tienen asignado un coste, el optimizador puede obtener el coste total asociado a cada posible plan, recorriendo el árbol del plan hacia arriba, propagando los costes desde las hojas hasta la raíz.

La manera de propagar los costes es calcular el coste asociado a cada nodo en función del coste asociado a sus hijos y del operador representado por el mismo (así como de la manera de ejecutarlo que contempla el plan: por ejemplo, join normal o anidado).

Una vez calculado el coste asociado a cada plan, el motor de ejecución pone en práctica el de menor coste asociado.

III.2.5.5. ALGORITMO PARA LA EJECUCIÓN DE UN MÉTODO DE BÚSQUEDA

Una vez que la vista asociada a la consulta ha sido obtenida, y el optimizador ha escogido el método de búsqueda de menor coste, el sistema tiene que obtener las tuplas de respuesta a la consulta.

Para ello basta recorrer recursivamente el árbol del método de búsqueda desde la raíz hasta las hojas, ejecutando el algoritmo indicado por cada tipo de nodo. Al llegar a los nodos hoja del árbol (que son las relaciones base), se habrá obtenido una sub-consulta sobre cada una de ellas. Estos resultados serán combinados para obtener la respuesta a la consulta.

```
Entradas: El nodo raíz del árbol del método de búsqueda
          Una lista de condiciones de consulta (vacía al comenzar, se
          irá rellenando según se atraviesen nodos selección)


Ejecutar (Nodo n, CondicionesConsulta conds[])
  {
  Si (N.tipo == union){

    // Lanzar (en paralelo)la consulta en los dos nodos hijo
    R1 = Ejecutar (N.hijoDerecho, conds)
    R2 = Ejecutar (N.hijoIzquierdo, conds)

    //Realizar la unión asíncrona de los resultados.
    Return R1 ∪ R2
  }

  Si (N.tipo == join) and (N.estrategia == convencional){

    // Dividir las condiciones de consulta. Se enviarán a cada nodo
hijo las condiciones de consulta referentes a sus atributos
    condsDerecha = escogerCondicionesSubrelacionDcha(condiciones)
    condsIzquierda = escogerCondicionesSubrelacionIzqda(condiciones)

    //Lanzar (en paralelo) la consulta en los dos nodos hijos
    R1 = Ejecutar (N.hijoDerecho, condsDerecha)
    R2 = Ejecutar (N.hijoIzquierdo, condsIzquierda)

    //Realizar el join asíncrono de los resultados.
    Return R1 ∞ R2
  }

  Si (N.tipo == join) and (N.estrategia == anidada){

    // Dividir las condiciones de consulta. Supongamos que el Nodo
derecho es el marcado como primero en el método de búsqueda
    condsDerecha = escogerCondicionesSubrelacionDcha(condiciones)
    condsIzquierda = escogerCondicionesSubrelacionIzqda(condiciones)

    //Lanzar la consulta en el nodo de la primera relación
    R1 = Ejecutar (N.hijoDerecho, condsDerecha)

    // Para cada n-upla única (a1i, a2i,..., ani)para los atributos de
join A1,...,An, de los obtenidos en R1, se envía la siguiente
consulta sobre la segunda subrelación (hemos supuesto que sería la
del nodo izquierdo)
```

```
    R2i= Select * from NodoIzqdo where (A1 = a1i) and ... and (An =ani)
and (condIzqda1) and ... and (CondIzqdaN)

    // Se calcula la unión de todos los resultados parciales
obtenidos de la segunda relación

    R2 = R2₁ ∪ R2₂ ∪ ... ∪ R2n

    //Finalmente, se calcula R1∞A1,...,AnR2 y se obtiene el resultado
final.
    Return R1∞A1,...,AnR2
  }

   Si (N.tipo == proyeccion){

    // Lanzar la consulta sobre el nodo hijo
    R1 = Ejecutar (N.hijo, conds)

    //Realizar la proyección para excluir los atributos ocultos
    Return Π (R1)
  }

  Si (N.tipo == seleccion){

    // Añadir las condiciones de la operación de selección
    conds = conds + condsSeleccion

    //Lanzar la consulta sobre el nodo hijo
    R1 = Ejecutar (N.hijo, conds)

    //Devolver los resultados
    Return R1
  }

  Si (N.tipo == hoja){

    // Lanzar la consulta al envoltorio
    R1 = InvocarEnvoltorio (conds)

    // Si un determinado atributo no aparece en la salida de la
consulta pero aparecía en la consulta con el operador '=', y un
determinado valor v, entonces el atributo se añade a los resultados
tomando para todas las tuplas el citado valor v
    CompletarResultados (R1)

    // Devolver los resultados
    Return R1
  }
}
Salidas: El conjunto de resultados para la consulta
```

III.2.5.6. SISTEMA DE CACHE

Como ya se comentó previamente, el mediador mantiene una cache que, en la implementación por defecto, es almacenada en una base de datos JDBC.

La cache es mantenida a nivel de relación. Esto quiere decir que la utilización o no de la cache (y su configuración asociada) puede decidirse individualmente para cada relación del sistema. Para cada una de las relaciones que utilice cache, el sistema generará una tabla en una base de datos JDBC.

Tres configuraciones habituales de funcionamiento son: *a nivel de relación base* (lo que quiere decir que sólo se guarda cache de las relaciones base), a *nivel global* (con lo que sólo se guardaría cache de las relaciones del esquema global) o a *nivel total* (se guarda cache de todas las relaciones, incluidas las intermedias de los árboles de vistas). Los dos primeros modos pueden funcionar a la vez y esa es, quizás, la configuración más común.

A medida que se vayan realizando consultas, las tablas se irán rellenando con las tuplas obtenidas. Nótese que si dos consultas devuelven resultados cuya intersección es no nula, las tuplas repetidas no se repetirán en la tabla de la base de datos JDBC, ya que las restricciones de unicidad del modelo relacional lo impedirán. De esta manera, consulta a consulta, se va construyendo una réplica de las relaciones base en el entorno local.

Además, cada una de estas relaciones mantendrá una descripción de las tuplas actualmente contenidas en cache, de manera que el mediador pueda saber qué consultas sobre la relación son solubles con datos de la cache.

III.2.5.6.1. DESCRIPCIÓN DE CONTENIDOS DE LA CACHE DE UNA RELACIÓN

En concreto una relación representará las tuplas que actualmente contiene en cache mediante una lista de *consultas patrón* que puede responder. Cada consulta patrón está formada por una lista de restricciones que deben ser satisfechas.

Cada restricción estará formada por un conjunto de tripletas de la forma `(Atributo Operador Valor)`

Si una consulta dada satisface todas las restricciones de una cualquiera de las consultas patrón, entonces puede ser resuelta por la cache.

Ejemplo III.2.5.2: Consideremos una determinada relación:

```
R={TITULO:String, AUTOR:String, PRECIO:Money},
```

representando los libros contenidos en una determinada tienda electrónica en Internet. A continuación se muestra una posible descripción de los contenidos de su cache.

```
{
  (TITULO Contiene 'java')
  (AUTOR Contiene 'Naughton')
}
```

Esta consulta patrón será verificada por cualquier consulta sobre R que incluya una consulta sobre al atributo TITULO con el operador Contiene por el valor java y que además incluya una consulta sobre al atributo AUTOR con el operador Contiene y por el valor 'Naughton'. Si estas dos restricciones son satisfechas, no hay restricciones sobre el resto de la consulta (siempre que los atributos adicionales por los que se consulte, estén a la salida de R).

Además, siempre que TITULO y AUTOR aparezcan en la salida de R, las consultas que utilicen operadores contenidos en aquellos representados en las restricciones, también serán validas. Por ejemplo, una consulta como:

```
Select * from R where (TITULO = 'java') and (Autor = 'Naughton')
```

también cumplirá las restricciones del patrón consulta anterior.

□

III.2.5.6.2. ALTA DE NUEVAS CONSULTA PATRÓN

Se creará una nueva consulta patrón asociada a una relación R cuando se reciba sobre ella una consulta que no pueda ser contestada por la cache, por lo cual el mediador irá a recoger nuevas tuplas desde la fuente.

La forma de crear un nuevo patrón partiendo de una consulta consiste, sencillamente, en extraer de la misma las tripletas que son implementadas buscando directamente en el formulario de la fuente, es decir, sin utilizar post-procesados.

Ejemplo III.2.5.3: Supongamos que una relación R tiene un único método de búsqueda de la siguiente forma:

```
{
 NEGATIVAS {
  (TITULO, CONTIENE, 1, CUALQUIERA)
 }
 POSITIVAS {
  (TITULO, CONTIENE, 1, CUALQUIERA)
  (AUTOR, CONTIENE, 1, CUALQUIERA)
 }
 SALIDA = {TITULO, AUTOR, PRECIO}
}
```

Si R recibe la consulta:

```
Select * from R where (TITULO = 'java') and (Autor = 'Naughton')
and(Precio <= 5000)
```

la implementará rellenando el campo TITULO del formulario de la tienda con el valor 'java' y el campo AUTOR con el valor 'Naughton'. Sobre los resultados obtenidos, aplicará un post-procesado para devolver sólo aquellos resultados que cumplan además la condición (Precio <= 5000). Sin embargo, a efectos de la cache, es preferible almacenar en la tabla PRECIO, ya que de esta manera se aumenta el número de consultas que la cache puede responder (en este caso concreto, podrá responderse a cualquier consulta por el atributo PRECIO siempre que la consulta contenga las condiciones (TITULO = 'java') y (Autor = 'Naughton').

□

En el momento de su creación, se asignará a la nueva consulta patrón un sello temporal con el momento de su creación. Esta información será utilizada para determinar cuando un determinado patrón consulta ha expirado.

De la misma manera, las tuplas que se hayan dado de alta en la tabla JDBC como efecto de la ejecución de la consulta que creó el nuevo patrón, se almacenarán junto con un sello temporal que permita eliminarlas cuando la consulta patrón haya expirado. Si una tupla ya estaba presente en la tabla, pero vuelve a recogerse desde la fuente como consecuencia de una nueva búsqueda, entonces su fecha de creación se actualiza a la más moderna (en la práctica, esta tupla ha sido *refrescada* y, por lo tanto, no tendrá que ser eliminada cuando expire su viejo patrón de consulta)

III.2.5.6.3. BAJA DE CONSULTAS PATRÓN

Cada relación tendrá un tiempo de expiración asignado. Cuando una consulta patrón expire, será eliminada.

Una consulta patrón también será eliminada cuando en la lista de consultas patrón se cree una consulta patrón menos restrictiva que ella.

Ejemplo III.2.5.4: Supongamos que la descripción de los contenidos de la cache de R contiene una consulta patrón como la siguiente:

```
{
  (TITULO Contiene 'java')
  (AUTOR Contiene 'Naughton')
}
```

Si R recibe la consulta:

```
Select * from R where (TITULO Contiene 'java')
```

habrá que crear una nueva consulta patrón de la siguiente forma:

```
{
  (TITULO Contiene 'java')
}
```

Es evidente que, dada la segunda consulta patrón, la primera es redundante y debe ser eliminada.

□

III.2.6. TRATANDO LA HETEROGENEIDAD DE LAS FUENTES EN LOS FORMATOS DE REPRESENTACIÓN DE LOS ATRIBUTOS

En el esquema mediado tendremos que trabajar con un esquema unificado y consistente, lo que incluye la representación de los diferentes valores de los atributos.

Sin embargo, para ello, tendremos que afrontar una serie de dificultades, que se ilustran en el siguiente ejemplo.

Ejemplo III.2.6.1: Heterogeneidades de representación

1) En ocasiones, fuentes diferentes pueden referirse a un mismo valor de un atributo de formas diferentes.

Una fuente conteniendo ofertas de empleo puede tener un atributo `categoría` en la que un posible valor sea `Construcción`. Sin embargo en el esquema mediado, o en otras fuentes, quizás ese mismo valor se denote como `Sector de construcciones`.

Otro ejemplo puede ser la representación de nombres propios en diversas fuentes. Una tienda electrónica de libros podría devolver valores para el atributo AUTOR con el formato 'Apellido, Nombre' ('Marías, Javier') mientras otra podría utilizar el formato 'Nombre Apellidos' ('Javier Marías').

2) También puede ocurrir que un determinado valor de un atributo del esquema mediado aparezca en una fuente desglosado en dos diferentes, cuya unión es semánticamente equivalente al valor deseado.

Siguiendo con el ejemplo anterior, en el esquema mediado podría existir un valor llamado `Telecomunicaciones e Informática`, mientras que en la fuente las ofertas de ambas disciplinas están separadas en dos valores diferentes: '`Ingeniería de Telecomunicaciones`' e '`Ingeniería Informática`'.

Recíprocamente, también puede ocurrir que en una fuente, un determinado valor de un atributo englobe varios valores del atributo en el esquema unificado. Continuando con el ejemplo, una fuente podría agrupar bajo el epígrafe `Cine` todas las ofertas relacionadas con la profesión cinematográfica. En el esquema mediado o en otras

fuentes, sin embargo, quizás se pretenda tener valores separados para `actores`, `directores`, etc.

3) También puede ocurrir que la información representada en un atributo, sea representada por una fuente usando dos o más atributos y viceversa.

Por ejemplo, el esquema puede utilizar un único atributo `categoría` para clasificar las ofertas de empleo, mientras que la fuente podría utilizar dos: `categoría` y `subcategoría`. También podría ocurrir el caso contrario: que el esquema mediado usase los dos atributos mencionados y la fuente utilizase sólo uno.
□

Es necesario, por lo tanto, proporcionar algún soporte para tratar estos casos. Este soporte debe proporcionarse tanto en el momento de la realización de consultas sobre las fuentes como en el momento de recuperación de los resultados desde las mismas.

Ejemplo III.2.6.2: Consideremos dos tiendas electrónicas de libros en Internet `A` y `B`. La tienda `A` utiliza para el campo `AUTOR` el formato de representación '`Apellidos, Nombre`'. La tienda B utiliza el formato '`Nombre Apellidos`'. Supongamos que se ha definido también una vista `R` = `A` ∪ `B`.

Consideremos un esquema mediado definido por la unión de ambas fuentes. Supóngase que deseamos que el formato utilizado en el campo `AUTOR` del esquema mediado sea '`Nombre Apellidos`'.

1) Supongamos que llega a `R` una consulta como:

```
Select * from R where (AUTOR ='Marias, Javier')
```

La consulta original debe dividirse en dos subconsultas, una para cada fuente. La subconsulta para `A` puede ser directamente la misma recibida en `R`. Sin embargo la consulta para `B` debe *reescribirse* y transformarse en la siguiente:

```
Select * from R where (AUTOR ='Javier Marias')
```

para ajustarse así al formato de representación de B. A esto es a lo que llamamos *tratar heterogeneidades en el momento de la consulta.*

2) Una vez realizadas las sub-consultas sobre `A` y `B`, el resultado para la consulta original se obtendrá uniendo los resultados de ambas subconsultas. Sin embargo las tuplas de `B` tendrán el campo `AUTOR` con el formato '`Apellidos, Nombre`'. Por lo tanto, antes de realizar la unión, es necesario reescribir dichos valores con el formato '`Nombre Apellidos`'. A esto es a lo que llamamos *tratar heterogeneidades en el momento de la recogida de resultados.*
□

III.2.6.1. TRATANDO HETEROGENEIDADES EN EL MOMENTO DE LA CONSULTA

Para poder tratar las heterogeneidades de una fuente, se proporciona un mecanismo que permite definir transformaciones (también llamadas *reescrituras*) en las consultas realizadas directamente sobre una fuente. En esta sección describimos brevemente sus ideas básicas

Recordemos que una consulta base sobre una determinada relación `R` en una fuente venía expresada por un conjunto de condiciones de consulta relacionadas por el operador `and`. Para componer una consulta pueden aplicarse los operadores selección, proyección, unión y join sobre las consultas base.

El mecanismo permite definir patrones representando una parte de una consulta que cumpla una serie de características, pudiendo asignar a cada patrón una acción cuyo resultado será un nuevo segmento de consulta.

Cuando llegue a la relación una consulta dentro de la cual algún subconjunto de sus condiciones, concuerde con alguno de los patrones definidos para esa relación, se invocará la acción asociada a dicho patrón y se sustituirá dicho segmento por aquel obtenido como resultado de la ejecución de la acción.

Ejemplo III.2.6.3: Reescritura de consultas

Definimos un campo `Categoría_Empleo` en la relación `G` del esquema global con los valores posibles: `"Sector de la Construcción"`, `" Telecomunicaciones e Informática"` y `"Actor"`.

Tenemos una fuente `E` que en el campo correspondiente tiene los valores: `"Telecomunicaciones"`, `"Informática"`, `"Cine"` y `"Construcción"`.

Podríamos definir los siguientes patrones y sustituciones:

```
1) {G, (CategoriaEmpleo = 'Telecomunicaciones e Informática')} >>>
{E, (CategoriaEmpleo = 'Telecomunicaciones')or (CategoriaEmpleo =
'Informática')}

2) {G, (CategoriaEmpleo = 'Sector de la Construcción')} >>> {E,
(CategoriaEmpleo = 'Construcción')}

3) {G, (CategoriaEmpleo contiene 'Actor')} >>> {E, (CategoriaEmpleo
contiene 'Cine') and (Descripción contieneor 'actor', 'actriz',
'actores', 'actrices', 'intérprete')}

(Se haría la búsqueda por el valor 'Cine' en la fuente. Se aplicaría
un filtrado consistente en quedarse sólo con los resultados que
```

```
contengan alguna de las palabras (operador contieneor): "actor,
actriz, actores, actrices, intérprete" dentro del campo 'Descripción'
devuelto en los resultados)
```

Supongamos ahora que la fuente tiene dos atributos llamados `categoría` y `subcategoría`, mientras que el esquema mediado tiene sólo un campo `categoría`. Para cada valor en el esquema mediado, asignaríamos en la fuente una consulta con el mismo significado semántico pero usando sus dos campos. Por ejemplo, si en la fuente tenemos una categoría `'Ingeniería'` y unas subcategorías llamadas `'Telecomunicaciones'` e `'Informática'`, mientras que en el esquema mediado tenemos un único campo y un único valor `'Telecomunicaciones e Informática'`, entonces para dicho valor podríamos definir la sustitución siguiente:

```
{G, (categoría = 'Informática y Telecomunicaciones')} >>> {E,
(categoría = 'Ingeniería') and (subcategoría = 'Telecomunicaciones')}
or {E, (categoría = 'Ingeniería') and (subcategoría = 'Informática')}
```

Recíprocamente, si la situación fuese a la inversa (dos atributos del esquema mediado se corresponden con uno sólo en la fuente), entonces para el ejemplo anterior, se podrían definir las siguientes sustituciones:

```
{G, (categoría = 'Ingeniería') and (subcategoría =
'Telecomunicaciones')} >>> {E, (categoría = 'Informática y
Telecomunicaciones') and(descripcion contieneor 'telecomunicaciones',
'teleco')}

{G, (categoría = 'Ingeniería') and (subcategoría = 'Informática')}
>>> {E, (categoría = 'Informática y Telecomunicaciones')
and(descripcion contieneor 'informática', 'programación','ciencias de
la computación')}
```

□

III.2.6.1.1. LENGUAJE DE DEFINICIÓN DE SUSTITUCIONES

Para ser capaces de definir transformaciones como las anteriores, se precisa de un lenguaje con la suficiente fuerza expresiva para representarlas. Además, dentro de lo posible, este lenguaje debe ser sencillo de utilizar y mantener una buena legibilidad.

III.2.6.1.1.1. Definición de patrones

Un patrón se constituirá de una o más *expresiones patrón*. Cada *expresión patrón* tendrá el siguiente formato:

```
{Relación,(conjunto_atributos,conjunto_operadores,
operación_concordancia (expresión_concordancia))}
```

donde:

- `Relación` especifica una relación.
- `conjunto_atributos` es una lista de atributos del mismo tipo de datos. Una tripleta incluida en una consulta concordará para este elemento si el atributo que especifica es uno de los contenidos en el conjunto.
- `conjunto_operadores` es una lista de operadores. Una tripleta incluida en una consulta concordará para este elemento si el operador que especifica es uno de los contenidos en el conjunto.
- `Operacion_concordancia(expresión_concordancia)`. `Operacion_concordancia` será una función definida sobre un tipo de datos, que recibiendo como parámetros una `expresion_concordancia` y un valor de ese tipo de datos, devuelva `verdadero` si el valor concuerda con la `expresión_concordancia` y devuelva `falso` en caso contrario. Por ejemplo, para el tipo de datos `String`, que representa atributos cuyos valores son cadenas de caracteres, podemos definir una operación que reciba como `expresion_concordancia`, una expresión regular y devuelva `verdadero` si el valor que se le pasa (una cadena de caracteres) concuerda con dicha expresión regular y `falso` en caso contrario. Se proporcionarán ya definidas funciones `Operación` para los principales tipos de datos y los usos más habituales. Podrán definirse más operaciones implementando una interfaz.

Además, diferentes *expresiones patrón* pueden combinarse utilizando los operadores proyección, unión y join para componer expresiones patrón que representen segmentos de consulta más amplios no restringidos a una sola relación.

III.2.6.1.1.2. Definición de operaciones de sustitución

Si un segmento `T` presente en una consulta `C` concuerda con alguno de los patrones definidos, entonces se ejecutará una *operación de transformación* que recibiendo `T` y `C` como parámetros, devolverá como salida un nuevo segmento `T'`. Sobre la consulta original, `T` será sustituido por `T'`. Las operaciones de transformación pueden ser implementadas por el administrador del sistema a conveniencia, garantizando así la extensibilidad. Se proporcionan funciones predefinidas para tratar con los casos más usuales.

III.2.6.1.2. OPERACIONES PREDEFINIDAS

El sistema proporciona ya definidas una manera de especificar serie de operaciones de sustitución, con sus correspondientes operaciones de concordancia que tienen la suficiente capacidad expresiva para ser útiles en la mayor parte de los casos.

Se propone a continuación un mecanismo predefinido para tratar uno de los casos más frecuentes: reformatear elementos de tipo `String` (cadenas de caracteres) de acuerdo a patrones expresados mediante expresiones regulares.

La operación de concordancia será en este caso un intérprete de expresiones regulares escritas en el formato utilizado en el lenguaje Perl[Per99]. La expresión recibida por dicha operación será por tanto una expresión regular del citado tipo. A cada grupo de la citada expresión regular (entendiendo "grupo" tal y como se entiende en la terminología de expresiones regulares Perl[Per99]) se le asignará un nombre mediante el mecanismo de prefijar dicho nombre con un signo '$'.

De esta manera, para definir la operación de transformación, se proporcionará una cadena textual construida a partir de los grupos anteriores.

Ejemplo III.2.6.4: Supongamos que tenemos un esquema global G con un determinado atributo NOMBRE que representa el nombre y apellidos de una persona con el siguiente formato 'Apellidos, Nombre' (e.g. 'Rodríguez, Juan'). Supongamos que una de las fuentes E tiene un atributo equivalente, pero que en ese caso se representa según el formato 'Nombre Apellidos' (e.g. 'Juan Rodríguez').

Para que las consultas sobre el esquema general se tradujesen correctamente a la fuente podríamos definir el siguiente patrón:

```
{G, (Nombre = ExprPerl ( ($APELLIDOS), ($NOMBRE)))}
```

y asignarle la siguiente transformación:

```
OperTransformacionExpr ({E, ($NOMBRE)  ($APELLIDOS)})
(notar el espacio en blanco que separa  ($NOMBRE) y  ($APELLIDOS))
```

□

III.2.6.2. TRATANDO HETEROGENEIDADES EN EL MOMENTO DE LA RECOGIDA DE RESULTADOS

De la misma manera que es necesario reescribir las consultas para tratar las heterogeneidades de representación en las fuentes, puede ser necesario transformar los resultados recibidos desde ellas para que su representación se adecue al formato del esquema global.

Este problema puede tratarse de manera similar al anterior pero utilizando las expresiones patrón para detectar tuplas devueltas por la fuente que concuerden con las mismas, y siendo la salida de las operaciones de transformación, los valores que realmente se devuelves como resultado de la consulta en la fuente.

III.2.7. DICCIONARIO DE DATOS

Como ya se ha comentado, el diccionario de datos de un sistema mediador debe presentar una estructura diferente de la de un diccionario de datos tradicional ya que

tiene que extenderse para almacenar las definiciones del esquema mediado, las capacidades de las fuentes, reglas para tratar la heterogeneidad de formatos etc.

Los aspectos relativos al nivel lógico están organizados dentro del catálogo de la siguiente manera:

- El diccionario de datos está organizado en distintas 'Bases de Datos'.
- Cada Base de datos tiene un esquema global.
- El esquema de una Base de Datos está compuesto por un conjunto de relaciones "virtuales".
- Una relación atómica está compuesta por un conjunto de *atributos*, que pertenecen, cada uno de ellos, a un determinado *tipo de datos* y pueden tener asociadas una serie de *restricciones*.
- Un *tipo de datos* lleva asociadas implícitamente una serie de restricciones de integridad que no se representan explícitamente en el diccionario de datos, dado que son impuestas por la implementación del tipo. Sin embargo, sobre los atributos de un determinado *tipo de datos* pueden ejecutarse una serie de *operadores*. Es necesario que el diccionario de datos almacene qué *operadores* pueden utilizarse con cada *tipo de datos*.
- Una *restricción* lleva asociada una implementación que actúa sobre un determinado atributo, para garantizar que los valores asignados a dicho atributo cumplen una condición determinada (en caso contrario, la tupla quedará descartada de la relación). Ejemplos de restricciones son NOT_NULL (que fuerza que dicho atributo no pueda tomar jamás el valor nulo) o UNIQUE (que fuerza que no pueda haber valores repetidos para el atributo.
- Cada relación del esquema puede o bien ser una relación base (representando a una fuente como tal) o venir definida por una *expresión en álgebra relacional*, expresada en función de otras relaciones (sería, pues, una relación derivada).
- Para cada relación (ya sea base o no) se almacenarán sus *capacidades de consulta*. Ya se ha visto como pueden representarse dichas capacidades, mediante el concepto de *método de búsqueda*.
- Para cada relación se almacenará su *información de costes*, que se utilizará para propósitos de optimización.
- Se almacenará también para cada relación, las posibles reglas que permitan tratar con heterogeneidades de esquema o formato y que sea necesario realizar para que la fuente sea integrable en el esquema global. La forma que toman esas reglas se ha visto ya.
- Para cada relación base, para cada método de búsqueda, se almacenará la información relativa a como implementar cada método de búsqueda en la fuente, así como la información de costes asociada a dicho método. Esta información es la utilizada por los *programas envoltorio* o *wrappers* para implementar el *nivel físico*.

III.3. GENERACIÓN SEMI-AUTOMÁTICA DE PROGRAMAS ENVOLTORIO PARA EL NIVEL FÍSICO

Como ya se ha comentado previamente, para garantizar la escalabilidad en número de fuentes de los sistemas basados en mediadores, es primordial que exista algún mecanismo que ayude a generar rápidamente los tipos más comunes de *programas envoltorio*.

En este apartado nos centraremos en como generar de manera semi-automática *envoltorios* para los siguientes tipos de fuentes:

- Fuentes JDBC. JDBC[JDB01] es la norma más apoyada por la industria para el acceso a Bases de Datos Relacionales desde el lenguaje JAVA. Esta norma es independiente del gestor de bases de datos utilizado y actualmente están soportados todos aquellos con una presencia significativa en el mercado. Por lo tanto, disponer de una manera rápida de generar *envoltorios* JDBC permite integrar en el sistema prácticamente cualquier tabla contenida en una base de datos relacional.
- Fuentes XML. XML[XML00] se está convirtiendo a pasos agigantados en la norma *de facto* para intercambio de datos. Muchas fuentes de información son capaces ya de exportar su información en dicho formato. Incluso aquellas que aún no lo hacen, pueden encontrar más sencillo hacerlo en este formato que en otros, dado el amplio abanico de herramientas disponibles para facilitar ese proceso.
- Fuentes de texto semi-estructurado (e.g. Fuentes Web). Incluye aquellas fuentes que, sin seguir un esquema rígido como el de una base de datos, guardan una cierta estructura. Esto incluye cualquier listado o tabla de items, compuesto cada uno de ellos por una serie de campos, separados de alguna manera. Ejemplos concretos pueden ser el listado de resultados de una búsqueda realizada en una fuente web o una lista de productos contenida en un documento escrito con algún procesador de textos comercial.

Un *envoltorio* debe realizar, principalmente, las siguientes tareas:

1. Representación de las capacidades de consulta de una fuente. Tal y como se comentó en apartados anteriores, las capacidades de consulta de una fuente pueden estar limitadas. El envoltorio debe ser capaz de representar dichas capacidades y de saber si una determinada consulta emitida sobre la fuente en el lenguaje general de consultas del mediador puede ser respondida o no por esta.

2. Representación de los formatos de salida de las consultas. El envoltorio debe representar el formato en el que devolverá al mediador los resultados de las consultas efectuadas sobre la fuente. Esto incluye los atributos disponibles en la fuente, sus tipos de datos, posibles restricciones sobre los contenidos, etc.

3. Obtener y exportar información de costes. Para propósitos de optimización, es importante disponer de información de costes sobre las consultas efectuadas

sobre las fuentes. Las labores de los envoltorios pueden incluir obtener esta información (quizás por algún proceso estadístico sobre el comportamiento de la fuente en consultas previas, tal y como se vio en el apartado II.5.2.2)

4. Traducir las consultas recibidas del mediador (y expresadas, por lo tanto, en el lenguaje de consultas del mismo) a el lenguaje de consultas de la fuente y ejecutar dicha consulta sobre la misma. Esto puede involucrar un proceso relativamente complejo que permita realizar sobre la fuente las operaciones precisas para obtener un documento o conjunto de documentos que contengan las respuestas a la consulta efectuada. Por ejemplo, en fuentes web, puede involucrar la realización de un proceso de navegación automática a través del sitio web y el rellenado automático de uno o más formularios HTML.

5. Del documento o conjunto de documentos obtenidos en la tarea 4), extraer las tuplas que constituyen la respuesta a la consulta. Los documentos que contienen las tuplas a extraer pueden además estar organizados en forma de árbol o grafo y el envoltorio puede tener que ser capaz de navegar a través de ellos. Por ejemplo, en una fuente web, las tuplas a extraer pueden estar distribuidas en diferentes documentos accesibles entre sí a través de hiperenlaces.

Para tratar los puntos 1) y 2), ya se ha visto que se proporciona un lenguaje declarativo que permite definirlos muy fácilmente. En cuanto al punto 3), si bien no forma parte de las aportaciones de este trabajo, ya se ha visto que los envoltorios del sistema mantienen estadísticas de costes de las consultas previas.

Sin duda, los puntos para los que es más importante proporcionar algún soporte de automatización o semi-automatización son el 4) y el 5) ya que en caso contrario, cada nuevo *envoltorio* requeriría la realización de código JAVA *ad-hoc*, labor muy costosa debido al tiempo necesario y al alto perfil de personal precisado para su realización.

En los siguientes epígrafes se tratan los aspectos involucrados en la generación semiautomática de *envoltorios* para cada uno de los tipos de fuente citados.

III.3.1. FUENTES JDBC

En una fuente JDBC, las *capacidades de consulta* implementadas por el *envoltorio* pueden expresarse mediante *métodos de búsqueda* descritos de la manera ya vista al tratar el nivel lógico.

Para convertir las capacidades de consulta expresadas por un método de búsqueda en consultas reales que puedan ser respondidas por una fuente JDBC, utilizaremos el concepto de *consulta patrón*.

Una *consulta patrón* es una consulta que puede ser parámetrizada en tiempo de ejecución. Cada método de búsqueda de una fuente JDBC tendrá asociada una *consulta patrón*.

En el caso más común, se deseará tener disponible dentro del mediador una tabla presente en una base de datos JDBC. Este tipo de *envoltorios* puede generarse con facilidad.

Ejemplo III.3.1.1: Envoltorio JDBC

Supongamos una tabla:

```
LIBRO={TITULO:String, AUTOR:String, EDITORIAL:String, PRECIO:Money}
```

contenida en una base de datos JDBC. Supongamos que definimos en un mediador una relación equivalente también llamada `LIBRO`. A dicha fuente asignamos un *método de búsqueda* único indicando que cualquier consulta puede realizarse sobre ella (ya que la fuente no impone ninguna restricción en las consultas a efectuar sobre ella). Podríamos definir la siguiente *consulta patrón* asociada a dicho método:

```
SELECT TITULO,AUTOR,EDITORIAL,PRECIO
FROM LIBRO
WHERE @EXPRWHERE (SubConsulta)
```

donde `@EXPRWHERE` es una función que se evaluará en tiempo de ejecución, que recibe como parámetro la subconsulta enviada a la fuente por el mediador (nivel lógico) y devuelve una expresión SQL equivalente, que es directamente insertable en la *consulta patrón*. Como quiera que el lenguaje nativo del nivel lógico del mediador es álgebra relacional, la conversión de las subconsultas a expresiones SQL directamente insertables en la *consulta patrón* es un proceso sencillo y fácilmente automatizable.

Los campos incluidos en el `SELECT` de la *consulta patrón* se harán corresponder directamente por su nombre a los atributos de la relación exportada.

□

A la hora de incluir una fuente JDBC, sin embargo, el creador de un sistema de integración puede desear no incluir directamente una tabla de una base de datos JDBC sino una vista sobre dicha base de datos. Para ello puede definir la vista utilizando la *consulta patrón* que desee.

Ejemplo III.3.1.2: Supóngase una base de datos JDBC, representando la cartelera de cine de una ciudad, que contiene tres tablas:

```
CINE={IDCINE:Long, NOMBRE:String, DIRECCIÓN:String, TELEFONO:String,
PRECIO:Money}
PELICULA={IDPELICULA:Long, TITULO:String, ACTORES:String,
DIRECTOR:String, ARGUMENTO:String}
CINE-PELICULA={IDCINE:Long, IDPELICULA:Long, HORARIO:String},
```

donde IDCINE es la clave primaria de CINE, IDPELICULA es la clave primaria de PELICULA y el par (IDCINE,IDPELICULA) es la clave primaria de CINE-PELICULA. Supongamos que deseamos que dicha fuente exporte al nivel lógico del mediador una relacion:

```
CARTELERA={NOMBRECINE, NOMBREPELICULA, PRECIO, HORARIO}
```

Podríamos definir la siguiente *consulta patrón*:

```
SELECT CINE.NOMBRE, PELICULA.NOMBRE, PRECIO, HORARIO
FROM CINE, PELICULA, CINEPELICULA
WHERE CINE.IDCINE=CINEPELICULA.IDCINE AND
CINEPELICULA.IDPELICULA=PELICULA.IDPELICULA AND
@EXPRWHERE(SubConsulta)
```

En cuanto a la *manera de implementar* el acceso a las tuplas de respuesta de la consulta y extraer las mismas, en este caso es muy sencillo realizar un programa *envoltorio* genérico que se conecta a cualquier base de datos JDBC, envíe la consulta obtenida tras la parametrización de la *consulta patrón* y obtenga las tuplas de respuesta. JDBC proporciona una interfaz totalmente normalizada para realizar estas tareas y sólo es preciso que dicho programa reciba como parámetros una clase *driver* JDBC para la base de datos que va a recibir la consulta y una URL hacia la misma en el formato especificado por JDBC (que indica la máquina, el puerto, el identificativo de usuario y la contraseña).

III.3.2. FUENTES DE TEXTO SEMI-ESTRUCTURADO Y FUENTES WEB

El aspecto más complicado en este tipo de fuentes es cómo convertir las capacidades expresadas por los métodos de búsqueda en consultas reales sobre la fuente.

Por un lado, es necesario expresar la manera de *realizar la consulta*, es decir de llegar a algún documento dentro del cual estén contenidas las tuplas que constituyan la respuesta a la consulta. Por ejemplo, en una fuente web, esto puede involucrar realizar una o varias peticiones HTTP[HTT97], que devolverán uno o varios documentos HTML[HTM99]. En el caso de otro tipo de fuentes, quizás sea necesario acceder a algún sistema de ficheros o enviar consultas a través de la *interfaz de programación* de alguna aplicación.

Por otra parte, una vez que el sistema ha llegado a dicho documento o documentos, es necesario extraer del mismo dichas tuplas, identificando además el valor para cada uno de sus atributos. Teniendo en cuenta que esta tarea debe poder realizarse sin desarrollar código *ad-hoc* para cada fuente, este es quizás el reto más complicado.

Esta dificultad deriva fundamentalmente de que la respuesta a las consultas viene embebida dentro de algún tipo de documento (e.g. una página HTML) que, a través de

una herramienta de visualización (e.g. un explorador de Internet) es "entendible para un humano", pero no así para un programa, que no recibe, en principio ninguna metainformación que le permita detectar en el documento las tuplas a extraer ni dividirlas en sus atributos constituyentes.

Por ejemplo, en una página web, las marcas del lenguaje HTML[HTM99] no dicen nada acerca del contenido de los datos mostrados (qué tipo de tuplas son, donde están, cuales son sus campos o atributos), simplemente proporcionan información de presentación para un explorador de Internet. Un programa que desee extraer las porciones de la página que representan las tuplas buscadas y estructurarlas en atributos deberá en principio disponer de alguna metainformación que le permita localizar las tuplas y dividirlas en sus campos.

La manera de realizar estas tareas en el sistema requiere una explicación amplia que se ofrece en el apartado III.4.

III.3.3. FUENTES XML

Las fuentes XML pueden ser consideradas como un subconjunto de las fuentes web o de texto semi-estructurado. Sin embargo, debido a su importancia particular, vale la pena distinguirlas y exponer algunas técnicas específicas para ellas.

En lo que se refiere a la manera de llegar al documento o documentos que contengan la respuesta a la consulta efectuada, no hay grandes diferencias.

En cuanto al proceso de extracción de dichas tuplas, es mucho más sencillo de automatizar, ya que los documentos XML, a través de sus marcas, identifican claramente las tuplas, así como sus atributos constituyentes. Existen además numerosas herramientas comerciales que permiten generar con facilidad analizadores para este tipo de documentos.

En concreto, el sistema mediador, partiendo de un DTD[XML00] especificando un lenguaje de marcas definido con XML, es capaz de generar de manera automática una relación base con una estructura que replique la del DTD y un envoltorio capaz de manejar transparentemente cualquier documento XML acorde al citado DTD.

También es permitido adoptar una idea similar a la utilizada para fuentes JDBC, pero donde las *consultas patrón* se expresen mediante el lenguaje de consultas XQuery[CFRS01].

III.4. WARGO: SISTEMA DE GENERACIÓN DE ENVOLTORIOS PARA FUENTES WEB Y DE TEXTO SEMI-ESTRUCTURADO

Una parte importante de la generación de envoltorios para fuentes semi-estructuradas consiste en extraer los resultados de una consulta dada, partiendo de un documento de información textual donde las tuplas están contenidas, pero sin que en dicho documento haya metainformación que permita localizarlas, ni identificar los atributos que las constituyen.

El ejemplo más clásico es el de fuentes web, donde los resultados para una consulta aparecen normalmente embebidos dentro de una página HTML[HTM99] sin que las marcas constituyentes de dicho lenguaje proporcionen información explícita sobre como localizar las tuplas e identificar sus atributos constituyentes.

Íntimamente ligado a este problema, especialmente en fuentes de hipertexto tales como las fuentes web, está el de cómo alcanzar el documento o documentos que contienen las tuplas resultado de la consulta (es decir, como *implementar* un método de búsqueda en la fuente). En fuentes web, esto normalmente involucra la realización de una o más peticiones HTTP, probablemente habiendo establecido previamente una sesión en la fuente, que debe mantenerse a lo largo de todo el proceso.

Un enfoque basado en realizar manualmente programas convencionales que realicen estas tareas de manera 'ad-hoc' para cada fuente web es demasiado costoso, ya que el tiempo y esfuerzo necesarios para crear y mantener dichos programas para un número de fuentes medio-alto puede ser enorme. Por un lado, realizar tales programas suele ser largo y tedioso. Por otro, esta tarea tiene que ser realizada por personal con conocimientos extensos de programación, lo cuál incrementa su coste.

Además, estos programas suelen ser muy dependientes de la estructura de la página de la que se quieren extraer las tuplas, por lo que prácticamente cualquier cambio en la misma obliga a reprogramar.

Por lo tanto, para casos con un número de fuentes medio-alto, es necesario disponer de algún sistema que permita generar dichos programas (a los que seguiremos llamando *envoltorios*, si bien se ocupan sólo de dos de las tareas que asignamos a los mismos en la sección III.3) de una manera más rápida y sencilla e intentando además disminuir la sensibilidad ante cambios en los formatos de las fuentes. Es también altamente deseable que este proceso sea lo suficientemente sencillo para ser realizado por personal sin conocimientos de programación. Incluso con un número bajo de fuentes, un sistema de este tipo puede disminuir significativamente los costes de creación y mantenimiento de mediadores.

En esta sección se describe un sistema que permite generar envoltorios para este tipo de fuentes, especialmente fuentes web, de manera mucho más rápida y sencilla que mediante programas ad-hoc, y que disminuye enormemente los costes de

implantación y mantenimiento de sistemas basados en mediadores. Además, esta tarea puede ser realizada por personal sin conocimientos de programación.

Este sistema recibe el nombre de WARGO[1], y tal y como veremos en la sección IV.2, ha sido utilizado para construir envoltorios para más de 600 fuentes diferentes, que están siendo actualmente utilizados en diversas aplicaciones industriales.

Wargo está especialmente orientado a la construcción de envoltorios para fuentes web, si bien, como veremos, también pueden abordarse otros formatos de representación de documentos distintos de HTML.

En fuentes web, para especificar cómo acceder al documento o documentos de los cuales se desea extraer datos, los usuarios pueden crear secuencias de navegación complejas por medio de ejemplos (es decir, realizándolas en su navegador), abstrayéndose de aspectos tales como Javascript, HTML dinámico o los mecanismos de mantenimiento de sesión ver sección II.5.4.1.1.1.

Para extraer la información requerida de los documentos obtenidos, Wargo proporciona una herramienta gráfica supervisora integrada en el navegador del usuario que le guía en la construcción de patrones de extracción de datos. El usuario debe marcar con su ratón ejemplos de los elementos de información a extraer y contestar preguntas simples realizadas por la herramienta a través de una interfaz gráfica sencilla. El resto de las tareas necesarias para generar el programa de extracción son asumidas por el sistema.

Wargo está construido sobre dos lenguajes de programación: NSEQL y DEXTL. Estos lenguajes y la herramienta supervisora que guía al usuario en la construcción de programas escritos en ellos, constituyen una de las principales aportaciones de esta tesis doctoral:

- NSEQL (Navigation SEQuence Language) es un lenguaje que permite definir secuencias de navegación tan complejas como se desee a través de fuentes web.
- DEXTL (Data Extraction Language) es un lenguaje para definir patrones de extracción de datos desde documentos de texto semi-estructurado (e.g. HTML). Es importante destacar que DEXTL no está limitado a funcionar exclusivamente sobre documentos HTML, sino que, en general, puede hacerlo sobre documentos escritos en cualquier lenguaje de marcas (como XML o XTG, el formato propietario de QuarkXPress), o de texto semi-estructurado.

En el siguiente apartado se proporciona una visión general de la arquitectura del sistema. Los siguientes apartados se ocupan, respectivamente, de describir NSEQL, DEXTL y de explicar cómo ambos lenguajes pueden combinarse entre sí. Finalmente,

[1] La tarea realizada por los envoltorios de fuentes web suele denominarse 'screen-scraping' o 'arañar la pantalla'. En la mitología creada por J.R.R. Tolkien, los wargos son lobos de afiladas garras, y por lo tanto con gran capacidad para 'arañar'.

se describen las ideas básicas sobre las que está construida la herramienta gráfica supervisora que permite al usuario generador de envoltorios abstraerse del conocimiento de los lenguajes presentados, cuando se estén generando envoltorios para fuentes web.

III.4.1. ARQUITECTURA DEL SISTEMA

Podemos distinguir dos etapas en el funcionamiento de Wargo:

- Tiempo de generación del envoltorio. Es la fase en la que el usuario utiliza la herramienta gráfica supervisora (o bien, si lo desease, directamente los lenguajes NSEQL y DEXTL) para la generación de la especificación de un envoltorio.
- Tiempo de ejecución. Es la fase en la que, partiendo de las especificaciones generadas en la fase anterior, el sistema ejecuta el envoltorio para realizar la extracción de datos.

Los siguientes dos sub-apartados exponen la arquitectura del sistema en cada una de estas fases.

III.4.1.1. ARQUITECTURA EN TIEMPO DE EJECUCIÓN

La figura III.4.1.1 describe l arquitectura del sistema en tiempo de ejecución.

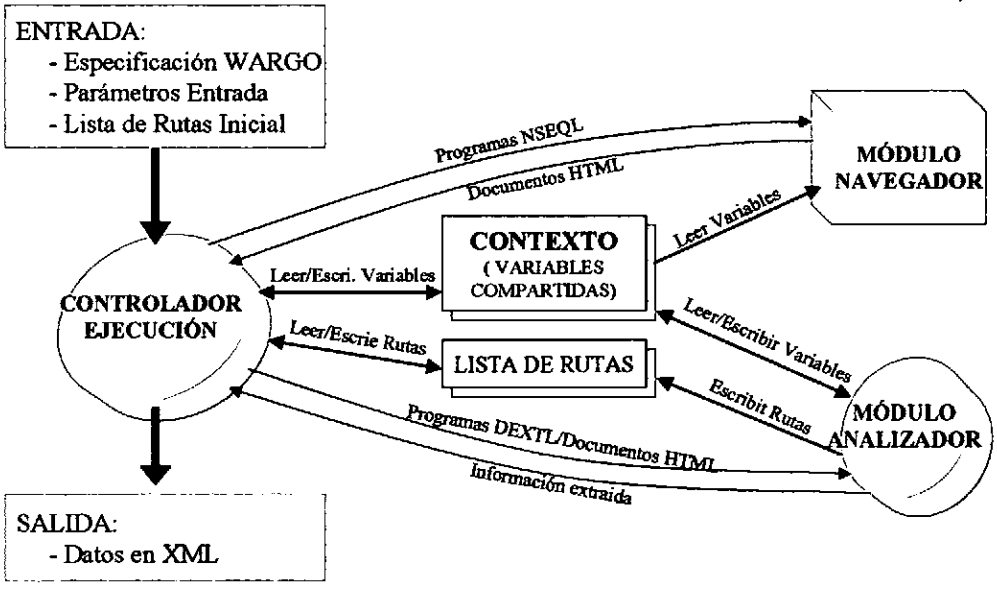


Figura III.4.1.1: Arquitectura en tiempo de ejecución

Como entrada de esta fase, Wargo recibe una *especificación de envoltorio* y una lista de *parámetros de entrada*:

- La *especificación de envoltorio* combina programas escritos en NSEQL y DEXTL para describir completamente las acciones necesarias para extraer los datos. Esta especificación es obtenida en tiempo de generación del envoltorio.
- Los *parámetros de entrada* son una lista de pares campo/valor que pueden parametrizar el comportamiento del envoltorio en tiempo de ejecución. Por ejemplo, un envoltorio construido para extraer una lista de mensajes desde un servidor de correo web, podría recibir dos parámetros `nombre_de_usuario` y `contraseña`, permitiendo así a la aplicación llamante especificar en tiempo de ejecución la cuenta de usuario de la cual deben extraerse los mensajes.

La *salida* de esta fase será un documento XML conteniendo los datos requeridos de la fuente. El esquema describiendo los datos de salida será obtenido también en tiempo de creación del envoltorio.

Los principales módulos del sistema en esta fase son:

- El *controlador* gestiona el proceso de ejecución. Interpreta la especificación del envoltorio e invoca al módulo *navegador* para ejecutar los programas NSEQL y al módulo *analizador* para ejecutar los programas DEXTL. También es responsable del proceso de exportación final de los datos obtenidos a XML.
- El módulo *navegador* es el encargado de interpretar y ejecutar los programas NSEQL, efectuando así las secuencias de navegación requeridas durante el proceso de extracción. En la implementación actual, este módulo está basado en el Microsoft Internet Explorer [IEX01].
- El módulo *analizador* es el encargado de interpretar y ejecutar los programas DEXTL sobre un documento dado, extrayendo del mismo los elementos de información requeridos. En la implementación actual, este módulo utiliza analizadores léxicos generados mediante Jflex [JFL01] y analizadores sintácticos obtenidos mediante un generador de analizadores sintácticos no deterministas de desarrollo propio.
- El *contexto* es un espacio de información compartido entre los diferentes módulos de Wargo que les permite intercambiar información entre sí, o guardar variables intermedias que puedan ser utilizadas en etapas posteriores del proceso de extracción. Más concretamente, tanto los programas escritos en NSEQL como los escritos en DEXTL pueden acceder a variables guardadas en este espacio. Además, los programas DEXTL pueden añadir nuevas variables al mismo dinámicamente. Los pares incluidos en la *lista de parámetros* recibida a la entrada se almacenan también como variables en el contexto.
- La *lista de rutas* contiene una lista de secuencias de navegación (normalmente expresadas en NSEQL) que apuntan a los documentos que el envoltorio debe explorar durante el proceso de extracción, junto con los programas DEXTL requeridos para extraer información de dichos documetnos. Típicamente, un envoltorio comienza conteniendo una única ruta en la lista y puede ir añadiendo

dinámicamente nuevas rutas si es necesario (tal y como se verá en apartados posteriores).

III.4.1.2. ARQUITECTURA EN TIEMPO DE GENERACIÓN DEL ENVOLTORIO

La Figura III.4.1.2 muestra la arquitectura de Wargo en tiempo de generación del envoltorio, cuando se utiliza la herramienta gráfica supervisora para fuentes web.

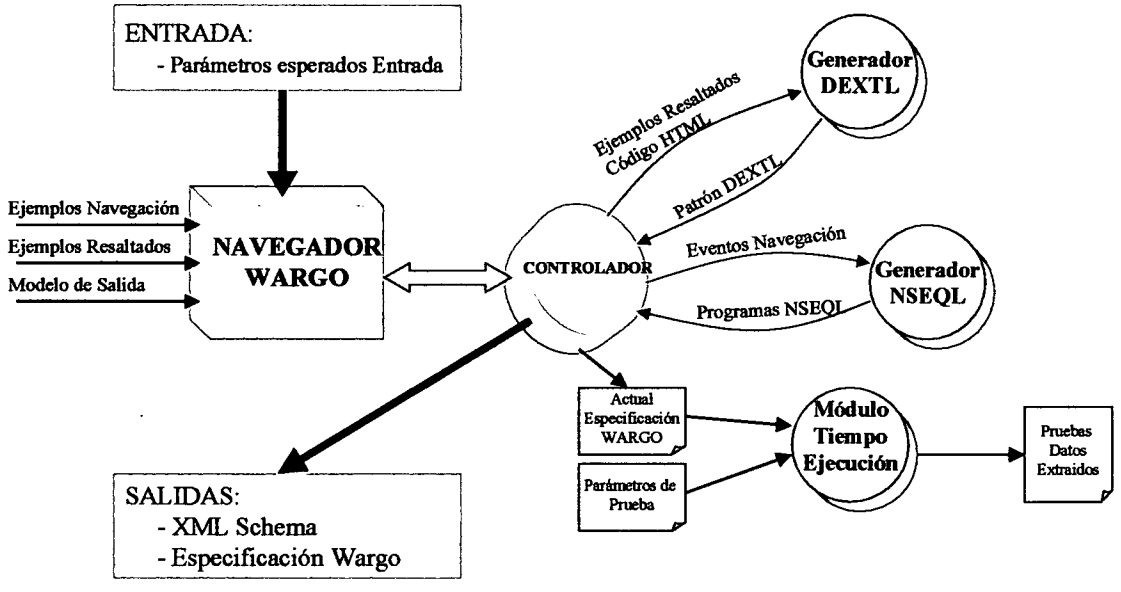


Figura III.4.1.2: Arquitectura de Wargo en tiempo de generación del envoltorio

La entrada para el proceso es una *lista de los parámetros esperados* posteriormente en tiempo de ejecución (es decir sólo el nombre de las variables, sin su valor).

La salida estará compuesta por una especificación que combinará programas escritos en NSEQL y DEXTL para la realización del proceso de extracción, así como de una descripción del esquema de los datos extraídos, mediante un *XML Schema* [XMS01].

Los principales módulos involucrados en esta fase son:

- El *navegador de Wargo* es una herramienta gráfica que guía al usuario durante el proceso de creación del envoltorio. Toma la forma de un navegador de Internet al que se le han añadido diversas opciones y funcionalidades específicas. La implementación actual está basada en Microsoft Internet Explorer.
- El *controlador* es el motor que implementa la lógica que rige el proceso. Utiliza el *generador NSEQL* para construir programas NSEQL que repliquen acciones de navegación realizadas por el usuario en el *navegador de Wargo* y el *generador*

DEXTL para construir programas DEXTL que extraigan los datos seleccionados por el usuario con su ratón, también a través de la interfaz proporcionada por el *navegador Wargo*.

- El *generador NSEQL* transforma los eventos de navegación generados por el usuario en el *navegador*, en programas NSEQL equivalentes. Es, por tanto, el motor básico que permite que el usuario pueda generar secuencias de navegación simplemente navegando.
- El *generador DEXTL* genera programas DEXTL partiendo de los ejemplos de la información a extraer marcados por el usuario en el *navegador*. Es por tanto, el núcleo que permite la generación supervisada de programas de extracción de datos.
- El *módulo de ejecución* (cuya arquitectura se comentó en el apartado anterior) es utilizado también en tiempo de creación del envoltorio, para probar y refinar las especificaciones construidas.

Finalmente, cabe decir que el uso de la herramienta de ayuda a la generación de envoltorios es opcional. Si lo desea, el usuario puede utilizar directamente los lenguajes DEXTL y NSEQL.

III.4.2. FUNDAMENTOS DE NSEQL

Una *ruta* es algún tipo de identificador que permite conocer la localización de un determinado documento. Por ejemplo, para documentos web que sean parte del web estático, la URL de un documento constituye una ruta hacia el mismo.

Si deseamos poder expresar rutas para cualquier documento web, ya sea estático o dinámico, precisamos usar un mecanismo de ruta más genérico que el que nos proporciona una URL. En concreto, podemos usar la abstracción de *petición HTTP*. Una petición HTTP puede especificarse mediante dos campos:

- *Método*. Puede tomar los valores GET o POST, que representan los dos métodos de invocación de formularios del mismo nombre, dentro del protocolo HTTP[HTT97].
- *URLPatron*. Es una manera de especificar una petición HTTP siguiendo la notación utilizada en HTML para las peticiones con el método GET y donde ciertas porciones de la petición pueden ser rellenadas en tiempo de ejecución, accediendo a los valores almacenados en el *contexto de navegación*. Estas porciones "sustituibles en tiempo de ejecución" se han prefijado con una '@'.

Ejemplo III.4.2.1: URL patrón

La URLPatrón:

```
http://www.sitioweb.com/cgi-bin/busquedalibros.jsp?
campotitulo=@TITULO&&campoautor=@AUTOR&&tiporesult=detallado
```

representa una petición HTTP a la máquina `www.sitioweb.com`, del CGI `/cgi-bin/busquedalibros.jsp`, pasándole a dicho CGI valores para los parámetros `campotitulo`, `campoautor` y `tiporesult`. El valor de `tiporesult` será siempre `detallado`. El valor concreto fijado para los parámetros `campotitulo` y `campoautor` puede variar en cada petición y será fijado en tiempo de ejecución de accediendo al *contexto de navegación*, en función de los identificadores que siguen al símbolo `@`. Esto quiere decir que, el valor determinado para el parámetro `campotitulo` será el valor asignado en el *contexto* para el identificador `TITULO` y el valor del parámetro `campoautor` será el valor asignado al identificador `AUTOR`.
□

Es conveniente resaltar que una URL estática puede expresarse mediante una petición `GET` de una URLPatron sin partes variables.

Sin embargo, tal y como se comentó en el apartado II.5.4.1.1.1, representar una ruta como una petición http e implementarlas con un cliente http tiene el importante inconveniente de que deben tratarse manualmente problemas como el tratamiento de Javascript y el mantenimiento de sesión en las fuentes que utilicen para ello identificadores de sesión.

Un enfoque más atractivo, que hasta donde sabemos ha sido utilizado hasta la fecha sólo en el sistema presentado en este trabajo, es utilizar una representación de más alto nivel que reproduzca los eventos generados por un usuario cuando se conecta a una fuente web utilizando un navegador de Internet (como Internet Explorer[IEX01] o Netscape-Mozilla[Moz01]). Esta es la idea básica detrás de NSEQL.

Ejemplo III.4.2.2: Secuencias de navegación con un explorador de Internet

Supongamos un sitio web que proporciona la funcionalidad de buscar libros a través de una interfaz que permite especificar el título y/o el autor de los mismos. El formulario es similar al expuesto en el ejemplo con peticiones http, con tres campos: `campotitulo` y `campoautor` para el título y el autor, y el campo oculto `tiporesult`.

Supóngase, sin embargo, que la fuente utiliza un esquema de mantenimiento de sesión basado en identificadores de sesión. El identificador de sesión se genera cuando el usuario se conecta a la página principal del sitio web y se arrastra a todas las demás páginas. Así, el formulario tendrá un campo adicional oculto, al que llamaremos `id_session`, en el que debe especificarse un identificador de sesión activo. En caso contrario, la consulta con el formulario no funcionará.

La única solución posible utilizando *rutas http* es conectarse a la página principal de la fuente, obtener manualmente con un código desarrollado ad-hoc, el número de sesión, y rellenar con ello el parámetro `id_session`. Este proceso, puede complicarse enormemente si, como es habitual en muchas fuentes comerciales, existen dificultades

adicionales como que el identificador esté dividido en varios fragmentos, que se encuentre escondido en marcos (o `frames`) ocultos, que en su composición se utilice código Javascript, etc. Además, si la fuente cambia su mecanismo de mantenimiento de sesión, el código ad-hoc debe reescribirse.

Sin embargo, es posible especificar esta navegación como la secuencia de acciones que un usuario realiza con su navegador para llegar a esa página y posteriormente, utilizar un navegador para implementarlas. NSEQL permite escribir tales secuencias de acciones.

Por ejemplo, en NSEQL, para este caso podríamos escribir:

```
1) NAVIGATE (http://sitioweb.com);
2) CLICKONANCHORBYTEXT ("Buscar libros",0);
3) FINDFORMBYNAME ("Búsqueda de Libros",0);
4) SETINPUTVALUE (campotitulo,0,"@TITULO");
5) SETINPUTVALUE (campoautor,0,"@AUTOR");
6) SUBMITFORM(0);
```

La línea 1 utiliza la función `NAVIGATE`, que recibe como parámetro una URL y se conecta a ella con el navegador de Internet. Para ello, no emite directamente ninguna petición http sino que su comportamiento es equivalente a que el usuario coloque la dirección en la barra de direcciones del navegador y pulse ENTER. El navegador responderá a la orden de 1) conectándose a la URL solicitada y construyendo el árbol DOM[HTM99] de la misma, que es la representación interna al navegador de todos los componentes de la página.

La línea 2 utiliza el comando `CLICKONANCHORBYTEXT(texto, n)` que genera un evento de tipo `click` sobre el `n`-avo enlace en la página cuyo texto coincida con el recibido (considerando que el valor `0` identifica al primer enlace). Esto causa que el navegador atraviese el enlace y se conecte a la página destino. NSEQL permite localizar enlaces de varias otras maneras alternativas, como por ejemplo, por su atributo `href`). Nuevamente, no se realiza directamente ninguna petición HTTP: el sistema navega por el árbol DOM del documento obtenido en el paso 1, hasta encontrar `n` nodos enlace con ese texto. Cuando lo encuentra, genera un evento de tipo "`click`" sobre dicho nodo, produciendo un efecto equivalente al producido por el usuario al pinchar con su ratón en el enlace. Por lo tanto, el sistema no precisa extraer el número de sesión: en general, podemos decir que todo aquello que es transparente para el usuario del navegador lo es también para el sistema.

La línea 3 utiliza la función `FINDFORMBYNAME(texto,n)`, que permite navegar por el árbol DOM hasta encontrar el `n`-avo formulario cuyo nombre sea el `texto` recibido (nuevamente, hay otras maneras disponibles para localizar un formulario, por ejemplo por el valor de su atributo `action`).

Las siguientes líneas (4 y 5) permiten asignar los valores deseados a los campos del formulario. Para ello se utiliza el comando `SETINPUTVALUE (campo, n, valor)`, que asigna `valor` al n-avo campo del formulario cuyo nombre es `campo`. En este caso se rellenan los campos `titulo` y `autor` con, respectivamente, los valores almacenados en las variables del contexto `TITULO` y `AUTOR` (nótese como las variables del contexto pueden ser utilizadas dentro de un programa NSEQL simplemente prefijándolas con el carácter '`@`').

Es importante notar que sólo es necesario rellenar aquellos campos visibles que el usuario también rellenaría. Así, el campo `tiporesult` es interno al sistema de la fuente web y, por tanto transparente al usuario, por lo que el sistema de navegación no tiene porque preocuparse por él. Por la misma razón, no necesita preocuparse del parámetro oculto en el que irá el identificador de sesión.

Finalmente, en la línea 5 se genera el evento `submit` sobre el nodo del árbol DOM que representa al formulario actualmente seleccionado, lo cual tiene un efecto equivalente a la acción de un usuario que pinchase con su ratón en el botón "`enviar`" de dicho formulario.

Como respuesta, se obtiene ya la página de respuesta que contiene las tuplas a extraer. Aunque no ocurre en este ejemplo, si la página de respuesta estuviese compuesta por varios marcos (o `frames`), podría seleccionarse uno de ellos con el comando `SelectFrame`, que permite seleccionar un marco por nombre, URL u orden en la página. NSEQL también incluye comandos para tratar otras características avanzadas como ventanas o menús emergentes.

Una selección de los principales comandos disponibles en NSEQL se muestra en la Figura III.4.2.1:

Command	Description
Navigate(url,cabeceras,destino)	Navega a "url" añadiendo las "cabeceras" especificadas (opcional) y carga el resultado en el marco "destino" (opcional).
PostData(url,datos,cabeceras,destino)	Envía por POST "datos" a "url" añadiendo las "cabeceras" especificadas (opcional) y carga el resultado en el marco "destino"
FindFrameByName (nombre,posición,igual)	Busca el n-avo marco llamado "nombre" (igual=true) o cuyo nombre contiene el "nombre" especificado (igual=false).
GetSourceCode()	Obtiene el código fuente del marco seleccionado actualmente
ClickOnAnchorByPosition(n)	Hace click en el n-avo enlace del documento.
ClickOnAnchorByText(texto,n,igual)	Hace click el n-avo enlace cuyo texto es igual a "texto" (igual=true) o cuyo texto contiene el "texto" especificado (igual=false).
FindFormByName(nombre,n)	Busca el n-avo formulario con el nombre" especificado.
SetInputValue(nombre,n,valor)	Establece el "valor" del n-avo campo de entrada del formulario activo, con el "nombre" especificado
SelectIndexByText(nombre,n,texto,n2,igual)	Selecciona la n2-ava opción con el texto especificado (igual=true) o conteniéndolo (igual=false), en el n-avo campo de tipo SELECT con el nombre especificado dentro del formulario activo
clickOnElement(nombre,tipo,n)	Hace click en el n-avo elemento del formulario activo, con el "nombre" especificado (opcional) y el "tipo" especificado.
FireEvent(nombreEvento,tipo,nombre,n)	Lanza el evento "nombreEvento" sobre el n-avo campo del "tipo" especificado y el "nombre" (opcional) especificado.
SubmitForm()	Envía el formulario activo.
FindElementByText(tipo,texto,n,igual)	Busca el n-avo elemento de tipo "tipo", con el texto "texto" (igual=true) o cuyo texto contiene el "texto" especificado (igual=false)
FindElementByAttribute(tipo,nombre,valor,n,igual)	Busca el n-avo elemento del "tipo" especificado, conteniendo un atributo con el "nombre" y "valor" especificados (igual=true) o conteniendo el "valor" especificado (igual=false)
ClickOnSelectedElement()	Hace click en el último elemento seleccionado
FireEventOnSelectedElement(nombreEvento)	Lanza el evento "nombreEvento" sobre el elemento seleccionado
OpenNewWindow(url,nombre,caracteristicas)	Abre una ventana hija con el "nombre" y "características" especificadas y carga la "url" especificada.

Figura III.4.2.1 Principales comandos de NSEQL

Especificar las secuencias de navegación a este nivel de abstracción no es solamente útil para evitar los procedimientos de establecimiento de sesión, sino también en el resto de situaciones ya comentadas en apartados previos: funciones Javascript, HTML dinámico, HTTPS, etc. Al igual que el usuario de un navegador no precisa lidiar con dichas interioridades del proceso de navegación, tampoco lo precisa el generador de envoltorios que utilice NSEQL.

Para fuentes con sistemas de establecimiento de sesión complejos y con uso abundante de Javascript (prácticamente todas las de banca electrónica o de agencias de viajes en línea, por ejemplo), nuestra experiencia dice que este sistema permite reducir los tiempos de generación de analizadores en más de un 50% y que reduce el nivel de cambios en las fuentes que afectan al analizador en más de un 25% (ver sección IV.2 para más detalle).

Por ello, si bien nuestro sistema soporta la definición de rutas tanto con el formato de petición http como con NSEQL, es altamente recomendada la utilización de NSEQL. La implementación actual trabaja sobre Microsoft Internet Explorer 5.5[IEX01] y superiores.

III.4.3. FUNDAMENTOS DE DEXTL

En esta sección se describen los fundamentos de DEXTL, un lenguaje para definir acciones de extracción de datos sobre documentos de texto semi-estructurado (típicamente, aunque no exclusivamente, HTML).

Los siguientes sub-apartados describen, respectivamente, el modelo de salida para los datos extraídos (sección III.4.3.1), las heurísticas sobre las que se basa el funcionamiento del sistema (sección III.4.3.2), y la descripción del lenguaje *DEXTL* en sí (sección III.4.3.3). Finalmente, se realiza una breve comparación con otros lenguajes de definición de programas de extracción de datos presentados con anterioridad en la literatura (sección III.4.3.4).

III.4.3.1. MODELO DE DATOS DE SALIDA

La salida de un envoltorio en Wargo es una relación compuesta por tuplas compuestas de atributos (a los que también llamaremos *elementos de información*), cada uno de ellos con un nombre. El contenido de un atributo puede ser o bien una cadena de caracteres, o bien una sub-relación con sus propios atributos.

Los atributos atómicos pueden tener asociado un *tipo de dato*. En DEXTL, los tipos de datos no son más que funciones booleanas que reciben una cadena de caracteres como parámetro y devuelven *verdadero* si la cadena representa una instancia válida del tipo de dato y *falso* en caso contrario. Por defecto, los atributos atómicos son considerados *cadenas* (que se representan mediante una función que siempre devuelve *verdadero*), si bien DEXTL incluye tipos pre-definidos tales como *fecha*, *dinero*, etc.

Ejemplo III.4.3.1: Modelo de relación en el sistema de extracción

Supongamos una relación R, que representa los libros en venta en una determinada librería electrónica en Internet. Para cada libro, la tienda ofrece la siguiente información: Título, relación de autores con el nombre de cada autor y relación de las diferentes ediciones del libro. Para cada una de las ediciones del libro se indican el formato, la editorial y el precio. Todos los atributos son de tipo *cadena* excepto el precio, que es de tipo *dinero*.

Podemos representar R de la siguiente forma:

```
R={TITULO,AUTOR{NOMBRE},EDICION{FORMATO,EDITORIAL,PRECIO:DINERO}}
```

□

Este modelo de salida de datos puede ser convertido directamente a un esquema XML [XMS01]. Los atributos que sean subrelaciones pueden ser considerados simplemente como elementos compuestos con cardinalidad '*'.

III.4.3.2. HEURÍSTICAS

En este contexto, una fuente, sin embargo, no devolverá los resultados a las consultas efectuadas en un formato estructurado, sino que los incluirá normalmente en un documento textual que, a través de su visualización con alguna herramienta, sea entendible y claro para un usuario humano.

Por ejemplo, en el caso más habitual de fuentes web, devolverá una página HTML que un navegador de Internet mostrará al usuario que realizó la consulta. Otro ejemplo puede ser el formato de documento XTG utilizado por herramientas como QuarkXPress [Qua02].

Por lo tanto, en el documento recibido como respuesta, habrá dos tipos de información claramente diferenciada:

- Textos que serán visualizados por el usuario (datos reales). En adelante los denominaremos *información para el usuario*.
- Textos y marcas con información de formato, que indican a la herramienta visualizadora cómo mostrar los datos (e.g. las marcas HTML que indican al navegador como mostrar los datos de la página). En adelante, las denominaremos como *información para la herramienta visualizadora*.

El sistema va a basar su funcionamiento en la asunción de ciertas heurísticas básicas de representación gráfica que los diseñadores de documentos (ya sea páginas HTML o de otro tipo) siguen para que sus documentos sean fácilmente legibles por los usuarios que los utilizarán.

Suponiendo el cumplimiento de dichas heurísticas, podrán extraerse una serie de conclusiones que permitirán automatizar partes del funcionamiento de los analizadores, haciendo así innecesario que esas tareas sean asumidas por el creador del analizador.

Para utilizar estas heurísticas, casi siempre será necesario "entender" el lenguaje de marcas utilizado para especificar la *información para la herramienta visualizadora*. Por ejemplo: ¿Cómo delimita un salto de línea el formato de representación del documento (e.g. HTML o XTG)?. Esto es debido a que, tal y como se verá, las heurísticas utilizadas se basan fuertemente en las maneras utilizadas habitualmente en los documentos textuales para separar visualmente items diferentes, tales como saltos de línea, tabuladores, etc.

Es importante notar que estas heurísticas delimitan el alcance del sistema. Esto es, si estas heurísticas no se cumplen, el sistema no será eficaz, ya que basa en ellas su funcionamiento interno.

Sin embargo, nuestra experiencia demuestra que las siguientes heurísticas son lo suficientemente generales para cumplirse en la inmensa mayoría de los casos (todas las

fuentes consideradas hasta el momento). El resto de sistemas mencionados en el estado del arte también asumen, implícita o explícitamente, estas heurísticas u otras similares.

III.4.3.2.1. HEURÍSTICA DE LISTADO HOMOGÉNEO

Esta heurística afirma que dos tuplas de una misma relación tienen una representación gráfica en la fuente igual o similar dentro de un mismo contexto. Esto quiere decir que la disposición de los diferentes atributos de la tupla dentro de su representación gráfica en la página de la que se extraen los datos, sigue un determinado patrón (o bien sigue uno de entre una lista de posibles patrones).

Ejemplo III.4.3.2: Heurística de listado homogéneo

Supongamos la salida de una búsqueda sobre una tienda electrónica de libros. La representación gráfica de los diferentes libros obtenidos como respuesta normalmente será muy similar, con la única diferencia entre un libro y otro, de que algunas partes de dicha representación pueden estar presentes en uno y en otro no. La Figura III.4.3.1 muestra un fragmento de la salida de una búsqueda por la palabra 'java' en la popular librería electrónica Barnes&Noble:

Beginning Java 2 - Jdk 1.3 Version
In Stock:Ships within 24 hours .
Ivor Horton / Paperback / Wrox Press, Inc. / March 2000
Our Price: $39.99, You Save 20%

The Complete Java 2 Certification Study Guide
In Stock:Ships within 24 hours .
Simon Roberts,Philip Heller / Hardcover / Sybex, Incorporated / September 2000
Our Price: $39.99

Figura III.4.3.1

Como puede verse, ambos libros son representados básicamente de la misma manera. Sin embargo, un examen minucioso mostrará que la representación no es exactamente igual. En concreto, el primer libro presenta en la última línea una información sobre descuentos del libro ("You save 20%") que no está presente en el segundo libro.

Esta leve heterogeneidad no causa el incumplimiento de la heurística, ya que la representación gráfica de los libros en esa página sigue siendo conforme a una lista de patrones alternativos (en este caso, habría dos patrones en esa lista, uno con la información de descuento y otro sin ella).

□

El fundamento de esta heurística es que, por motivos de claridad, en cualquier documento, elementos del mismo tipo suelen representarse de la misma manera. Por ejemplo, una tienda de libros, no representará cada libro resultado de una búsqueda de una manera totalmente diferente en cada ocasión ya que:

- Podría causar confusión en el usuario de la tienda
- La información mostrada es homogénea y estará probablemente almacenada en última instancia en algún tipo de base de datos, aunque esta no esté directamente accesible al usuario. Las páginas de resultado de la consulta serán normalmente generadas por un programa que ajustará su funcionamiento a dicha base de datos y que normalmente producirá una representación uniforme.

III.4.3.2.2. SEPARACIÓN HORIZONTAL

La separación entre una tupla de una relación y la siguiente estará marcada gráficamente de alguna manera. Es decir, existe algún delimitador que indica cuando termina una tupla y cuando comienza la siguiente.

La razón por la que esta heurística se cumple en la práctica es que, en caso contrario, la representación resultaría confusa para el usuario.

III.4.3.2.3. SEPARACIÓN VERTICAL

Los valores de los atributos de una tupla están separados o diferenciados visualmente de alguna manera.

Ejemplo III.4.3.3: Heurística de separación vertical (I)

En una tienda electrónica, para un determinado libro, la cadena que identifica al título del libro estará claramente separada gráficamente de la cadena que identifica el autor, de la que identifica el precio, etc. La separación puede realizarse a través de una cadena separadora (ej. Un '`;`' o un '`.`'), del hecho de que alguno de los campos esté resaltado por un enlace, del uso de una fuente de letra diferente, etc.

□

La razón por la que esta heurística se cumple en la práctica es que, en caso contrario, la representación resultaría confusa para el usuario.

Ejemplo III.4.3.4: Heurística de separación vertical (II)

Supongamos que una tienda electrónica no respetase esta heurística para el caso del título y el autor de un libro. Podría escribir algo del estilo: '`Corazón tan blanco Javier Marías`'.

Si bien es cierto que un aficionado a la lectura, haciendo uso de su conocimiento semántico sobre el dominio, podría diferenciar en este caso qué parte de la cadena

anterior es el título del libro y qué parte es el autor, no es menos cierto, que esta representación es confusa. Una representación que delimite de alguna manera qué parte de la cadena corresponde a cada campo es mucho más probable. Por ejemplo: 'Corazón tan Blanco. Javier Marías', donde el subrayado del título y el punto después del mismo separan claramente ambos atributos.

□

III.4.3.2.4. UNIDAD DE TUPLA

Los diversos campos de una misma tupla estarán asociados unívocamente, ya sea porque están en la misma página gráficamente relacionados, ya sea porque partiendo de alguno de los campos de la tupla se puede acceder al resto de ellos siguiendo algún tipo de hiperenlace o referencia que esté gráficamente asociado con él. Esto incluye también el caso de atributos constituidos por sub-relaciones.

Ejemplo III.4.3.5: Heurística de unidad de tupla

En la relación:

```
R={TITULO, AUTOR{NOMBRE}, EDICION{FORMATO, EDITORIAL,
PRECIO:DINERO}}
```

las tuplas de EDICION correspondientes a una tupla de R, estarán gráficamente ligadas de alguna manera a los valores para TITULO y AUTOR de dicha tupla. Es decir, la información sobre las ediciones de un libro determinado estarán asociadas gráficamente de alguna manera al resto de información sobre dicho libro.

□

La razón del cumplimiento de esta heurística es que, es que si no se cumpliese, el usuario no podría percibir la ligazón lógica que existe entre los campos de una misma tupla.

III.4.3.3. ESTRUCTURA DE UN PROGRAMA DEXTL

Un programa DEXTL está compuesto por elementos estructurados jerárquicamente. Los elementos en un programa DEXTL se corresponderán, típicamente, con los elementos de información que se desea extraer de un documento dado, por lo que la estructura de un programa DEXTL es similar al modelo de salida de datos expuesto en la sección III.4.3.1.

Por tanto, cada elemento DEXTL podrá ser *atómico* o *no atómico*, correspondiéndose los elementos no atómicos con los atributos de tipo sub-relación. Los elementos atómicos podrán tener asociado un tipo de dato (interpretado de la misma manera que en el modelo de salida).

La construcción básica sobre la que se construyen los programas DEXTL es llamada *patrón*. Un patrón permitirá extraer las ocurrencias de un determinado elemento de información *no-atómico* dentro de un documento, así como de todos sus sub-elementos. Los patrones DEXTL son detallados en la sección III.4.3.3.1

En la sección III.4.3.3.2 se muestran características del lenguaje precisas para construir programas DEXTL completos.

III.4.3.3.1. PATRONES DEXTL

Este apartado analiza en detalle la principal construcción de DEXTL. En primer lugar, se exponen los fundamentos de su construcción y de la manera en que son interpretados. Los siguientes sub-apartados tratan diversas características avanzadas: definición de marcas de formato, separadores avanzados, uso del contexto compartido e inclusión de acciones en los patrones.

III.4.3.3.1.1. Fundamentos básicos del sistema de patrones

El Generador de analizadores permite al usuario definir un patrón que especifique una representación gráfica de las tuplas de una determinada relación R en un determinado documento o conjunto de documentos (por ejemplo en las páginas de resultado de una búsqueda). La heurística de *Listado homogéneo* asegura que dicho patrón (o lista de patrones) existe.

Una vez definido dicho patrón, el sistema buscará sus ocurrencias en el documento o conjunto de documentos especificado.

Debe recordarse que un documento contiene dos tipos de información: *información para el usuario* e *información para la herramienta visualizadora*, que aparecen mezclados en el documento. Un patrón puede estar compuesto por elementos que representen información de ambos tipos, de la manera que se verá a continuación.

En primer lugar, decir que dentro de un programa DEXTL, un patrón irá siempre contenido dentro de un determinado *nivel*. Tal y como se verá en detalle más adelante, un nivel dentro de una especificación se corresponde con una relación o subrelación cuyas tuplas se quieren extraer a través de dicha especificación. Por ahora, basta tener en cuenta que:

- Cada nivel tiene asociado un nombre, que debe coincidir con el de la relación o subrelación cuyas tuplas extrae y que se indica con la notación `PATRON NOMBREPATRON`, donde `NOMBREPATRON` es el nombre asignado al nivel.
- La especificación de un nivel se delimita mediante el carácter de comienzo '`{`' y el carácter de fin '`}`'
- Dentro de la especificación de un patrón pueden incluirse líneas de comentarios, prefijadas con los símbolos '`//`'.

- Todo nivel contiene dentro de su especificación al menos un patrón. Cada patrón representa un aspecto gráfico en el documento de las tuplas a extraer.

Un patrón estará compuesto fundamentalmente por los siguientes tipos de elementos:

- Separadores. Sirven para que el sistema sea capaz de distinguir donde empieza o acaba, dentro de un patrón, un determinado atributo, así como donde empieza o termina una determinada tupla. La existencia de estos separadores, tomando la forma de algún tipo de representación gráfica, es garantizada por las heurísticas de *Separación Vertical* y *Separación Horizontal*. En un patrón habrá básicamente dos tipos de separadores:
  - Cadenas de caracteres. Concuerdan con porciones del documento representando *información para el usuario*. Se representan entrecomilladas en el patrón.
  - Marcas de formato. Se utilizan para representar separadores constituidos por alguna expresión en el lenguaje utilizado para representar la *información para la herramienta visualizadora*. Las marcas se definen normalmente para proporcionar una manera sencilla y legible de representar las estructuras visuales presentes en el documento, y que están expresadas en la *información para la herramienta visualizadora*. Por ejemplo, en la mayor parte de lenguajes de representación de documentos (HTML[HTM99], PDF[PDF00]) existen marcas que permiten especificar un retorno de carro (o 'fin de línea'). Puede ser muy útil definir una marca `FINLINEA` como una expresión regular que concuerde con dichas marcas, proporcionando así una manera simple y altamente legible de representar los retornos de carro en los patrones. Las *marcas de formato* se definen mediante expresiones regulares a las que se asigna un nombre. Es importante destacar que las porciones del documento representando información de formato que no concuerden con ninguna marca de formato, son ignoradas por el sistema.
- Nombres de Atributos en R. Se situarán en la posición del patrón en la que aparece el valor de dicho atributo dentro de la representación gráfica de las tuplas de R. Deben prefijarse con el carácter ':'. Concuerdan con las porciones del documento que representan *información para el usuario* (si bien hay una excepción a esto que se verá en el apartado III.4.3.3.1.2) y que se encuentran entre dos separadores. También es posible tener atributos auxiliares que no pertenezcan a R, pero cuyo valor sea interesante recoger para alguna operación o cálculo. Los atributos auxiliares se prefijan con '::'.
- Etiqueta `NOINTERESA`. Funciona de manera similar a los nombres de atributos en R, pero representa una porción del patrón que no se desea asignar a ningún atributo de R.
- Marcas de opcionalidad. Son "¿" y "?" y permiten delimitar zonas del patrón que pueden aparecer o no.
- Marcas para identificar porciones del patrón alternativas. En un patrón puede aparecer una expresión del estilo `(subpatron1 | subpatron2)`, indicando que

en esa zona del patrón puede aparecer o bien algo que concuerde con `subpatrón1` o bien algo que concuerde con `subpatrón2`.

Para encontrar las ocurrencias del patrón en la página, el sistema precisará de la intervención de dos módulos: un analizador léxico y un analizador sintáctico.

El analizador léxico tiene como misión "trocear" el documento de entrada en 'tokens'. En este caso, los tokens podrán ser *marcas de formato* o textos que sean parte de la *información para el usuario* contenida en el documento.

El analizador sintáctico recibe los tokens que le envía el analizador léxico y detecta las ocurrencias del patrón buscado.

El analizador léxico se comportará de la siguiente manera:

- El sistema ignorará toda la *información para la herramienta de visualización* contenida en el documento excepto aquellas porciones que encajen con la definición de alguna de las *marcas de formato* presentes en el juego de marcas utilizado.
- Aquellas porciones de la información de formato que encajen con alguna de las marcas de formato, se identifican como tales y se devolverán al analizador sintáctico.
- Se obtienen los textos del documento visibles para el usuario (es decir, en un documento HTML, aquellos que no van entre '<' y '>'). La división en 'tokens' de los textos contenidos en la *información para el usuario* se realiza por el sencillo procedimiento de "cortar" el texto cada vez que aparece en el documento una marca de formato

Ejemplo III.4.3.6: Fundamentos de los patrones DEXTL (I)

Supongamos una fuente web con información sobre libros, modelada mediante una relación `R={TITULO, AUTOR}`. Supongamos que el siguiente fragmento de HTML representa la información que la fuente ofrece sobre un determinado libro:

```
<br> Michael <b> Crichton </b> <br>
<a href="/cgibook.jsp?id=1256790">Title: Jurassic Park </a><br>
```

Para poder representar un retorno de carro en el patrón, se define una marca `FINLINEA` como una expresión regular que comprenda todas las posibles maneras que hay en HTML de representar un retorno de carro o fin de línea (como, por ejemplo, usando la marca `<br>`). Si bien, hay más maneras de representar un retorno de carro en HTML (ver Figura III.4.3.3 para una definición exhaustiva), una definición simple para `FINLINEA`, útil en muchos casos, podría ser la siguiente:

```
FINLINEA= ("<br>"|"</p>"|"</tr>")
```

En la definición superior, se asocia la marca FINLINEA con una expresión regular que concordará con la *información para la herramienta visualizadora* contenida en el documento, cada vez que aparezca o bien la marca
 o bien la marca </p> o bien la marca </tr>.

En esas condiciones, el analizador léxico devolverá los siguientes tokens al analizador sintáctico:

1) Token de tipo FINLINEA: Correspondiente al primer
.
2) Token de tipo TEXTO, conteniendo el valor 'Michael Crichton'. Nótese que las marcas y son ignoradas a todos los efectos, ya que constituyen *información para la herramienta de visualización* que no concuerda con ninguna de las marcas de formato definidas.
3) Token de tipo FINLINEA. Correspondiente al segundo
.
4) Token de tipo TEXTO, conteniendo el valor 'Title: Jurassic Park'. Nótese que las marcas y son ignoradas a todos los efectos, ya que no concuerdan con ninguna de las marcas de formato definidas.
5) Token de tipo FINLINEA. Correspondiente al tercer
.

Para entender como funciona el analizador sintáctico, supóngase ahora que se ha definido el siguiente patrón:

```
PATRON R
MARCAS {FINLINEA}
{
 :AUTOR FINLINEA "Title:" :TITULO FINLINEA
}
```

Dado este patrón, el analizador sintáctico detectará una ocurrencia del patrón cuando reciba consecutivamente del analizador léxico la siguiente secuencia de tokens: TEXTO, FINLINEA, TEXTO, FINLINEA. Además, el segundo token TEXTO tiene que cumplir una restricción adicional: debe ser conforme a la expresión regular que va implícita en la porción del patrón ["Title:" TITULO], y que en este caso, determina que el valor asociado a dicho token debe comenzar por la cadena "Title:".

Como puede verse, para el extracto de código HTML anterior, los cuatro últimos tokens devueltos por el analizador léxico constituyen una secuencia que concuerda con el patrón. Una vez detectada dicha secuencia de tokens, las acciones realizadas por el analizador sintáctico serán las siguientes:

- Crear una nueva tupla de la relación.
- Asignar el valor del primer token TEXTO al campo AUTOR de la nueva tupla. Para nuestro ejemplo, el valor que tomará el campo AUTOR será 'Michael Crichton'.

- Asignar al campo TITULO el resultado de aplicar al valor del segundo token TEXTO, la expresión regular construida implícitamente mediante el uso de separadores del tipo cadena de caracteres, que consiste en nuestro caso en suprimir la subcadena 'Title:'. Por lo tanto, para nuestra tupla ejemplo, el valor para el campo TITULO será 'Jurassic Park'.

Una vez vistos los fundamentos básicos, considérese el siguiente ejemplo, de mayor complejidad.

Ejemplo III.4.3.7: Fundamentos de los patrones DEXTL (II)

Consideremos el ejemplo de la Figura III.4.3.1, que representa el aspecto en un navegador de Internet de la página HTML devuelta como resultado de una búsqueda en una tienda online de libros. En la Figura III.4.3.2, se muestra el código HTML asociado a la representación del primer libro.

Para representar con comodidad patrones capaz de extraer datos desde páginas HTML podemos pensar en definir una serie de *marcas de formato*, para representar las maneras habituales de separación mediante tags en HTML. Por ejemplo, viendo la Figura III.4.3.1 podemos ver que en este caso los diferentes campos de cada tupla están separados o bien por un retorno de carro o bien por algún tipo de cadena de caracteres.

Por lo tanto, podemos definir la marca de formato FINLINEA, de la misma manera que en el ejemplo anterior, pudiendo así representar en el patrón los retornos de carro.

```
<font size="-1" face="arial, helvetica, sans-serif">
<A
href="/booksearch/isbnInquiry.asp?userid=2N3SK9Y2P7&mscssid=JF86DUA
3QD498J283583M4SBG9PQ0UL7&isbn=1861003668">
<b>Beginning Java 2 - Jdk 1.3 Version</b>
</a></font><br>
<font size="-1" face="arial, helvetica, sans-serif">
<font color="#aa0000">In Stock:Ships within 24 hours
.
</font><br>Ivor Horton / Paperback / Wrox Press, Inc. /
<font size="-1" face="arial, helvetica, sans-
serif">March 2000<br>
Our Price: <font color="#aa0000">$39.99</font>, You Save <font
color="#aa0000">20%</font><br>
</font></TD>
```

Figura III.4.3.2: Código HTML de uno de los resultados de la Figura III.4.3.1

En esas condiciones, un patrón para extraer la información de las tuplas de una relación R={TITULO,AUTOR,FORMATO,PRECIO,DESCUENTO} representando una

tienda en Internet que utilice el formato de resultados mostrado en la Figura III.4.3.1, con el código HTML de la Figura III.4.3.2, podría expresarse de la siguiente manera:

```
PATRON R
MARCAS {FINLINEA}
{
 :TITULO FINLINEA
 NOINTERESA FINLINEA
 :AUTOR "/" :FORMATO "/" NOINTERESA FINLINEA
 "Our Price:" :PRECIO ¿",You Save" :DESCUENTO FINLINEA?
}
```

Nótese el uso de la etiqueta NOINTERESA para representar aquellas porciones de la *información para el usuario* que no se desean asignar a ningún atributo de la tupla. Nótese también el uso de las marcas de interrogación "¿" y "?" para delimitar una porción del patrón opcional (esto es, que aparece sólo para algunos resultados de la búsqueda).

Una vez definido el patrón de las tuplas, el sistema buscará las ocurrencias de ese patrón en la página. Si consideramos el código HTML de la Figura III.4.3.2, y recordando que aquellas partes de la *información para la herramienta de visualización* que no concuerden con ninguna marca de formato, pueden eliminarse, podemos obtener el siguiente código HTML, que será el que realmente considere el sistema:

```
Beginning Java 2 - Jdk 1.3 Version <br>
In Stock:Ships within 24 hours.<br>
Ivor Horton / Paperback / Wrox Press, Inc. / March 2000<br>
Our Price: $39.99, You Save 20%<br>
```

De esta manera, el analizador léxico devolverá sucesivamente los siguientes tokens al analizador sintáctico:

- Token de tipo TEXTO con el valor asociado 'Beginning Java 2 - Jdk 1.3 Version'.
- Token de tipo FINLINEA, correspondiente con el primer
.
- Token de tipo TEXTO, con el valor asociado 'In Stock:Ships within 24 hours.'
- Token de tipo FINLINEA, correspondiente con el segundo
.
- Token de tipo TEXTO, con el valor asociado 'Ivor Horton / Paperback / Wrox Press, Inc. / March 2000'
- Token de tipo FINLINEA, correspondiente con el tercer
.
- Token de tipo TEXTO, con el valor asociado 'Our Price: $39.99,You Save 20%'
- Token de tipo FINLINEA, correspondiente con el último
.

Dada esta secuencia de tokens, el analizador sintáctico reconocerá una secuencia válida del patrón y realizará las siguientes acciones:

- Crear una nueva tupla de R.
- Asignar al campo TITULO de la nueva tupla el valor asociado al primer token de texto, esto es 'Beginning Java 2 - Jdk 1.3 Version'.
- Descartar el valor del segundo token de tipo TEXTO (ya que se corresponde con un elemento NOINTERESA).
- Dividir el valor asociado al tercer token de tipo TEXTO de acuerdo a la expresión regular implícita en el patrón. En concreto, se dividirá en tres partes delimitadas por el carácter separador "/". La primera porción se asigna como valor del campo AUTOR de la tupla, la segunda porción se asigna al campo FORMATO, la tercera se descarta ya que se corresponde con un elemento NOINTERESA. De esta manera el campo AUTOR de la tupla toma el valor 'Ivor Horton' y el campo FORMATO toma el valor 'Paperback'.
- Dividir el valor asociado al cuarto token de tipo TEXTO en fragmentos delimitados por los separadores 'Our Price:' y ', You Save'. El fragmento que se encuentra entre estos dos separadores se asigna al campo PRECIO de la tupla. El fragmento que se encuentra entre el separador ', You Save' y el final del texto se asigna al campo DESCUENTO de la tupla. De esta manera, el campo PRECIO toma el valor '$39.99' y el campo DESCUENTO toma el valor '20%'.

□

III.4.3.3.1.2. Definición de Marcas de Formato

La definición de *marcas de formato* suele ser realizada por administradores del sistema conocedores del lenguaje de marcas utilizado en la *información para la herramienta visualizadora*.

DEXTL incluye ya definidas marcas de formato que representan los principales elementos gráficos que pueden encontrarse en HTML. De esta manera, el programador de DEXTL precisa tan sólo escoger las que desea utilizar para cada patrón, de entre las definidas.

Cada patrón DEXTL puede especificar las marcas a utilizar poniendo en la primera línea después del nombre del patrón, la expresión MARCAS={MARCA1,...,MARCAn}, donde MARCA1,...,MARCAn representan las marcas de formato que se desea utilizar. Si para un patrón no se especifica juego de marcas, se asume que utilizará el conjunto que se haya especificado por defecto (para documentos HTML, el conjunto seleccionado por defecto suele contener los separadores ENLACE, FINLINEA y TABULADOR).

La figura III.4.3.3 muestra algunas de las marcas de formato más comúnmente utilizadas para documentos escritos en el lenguaje HTML.

Marca Formato	Definición	Descripción
FINLINEA	"</TD>" [\n\r\t]* "</TR>" | " " | "<LI>" | "<P"~">" | "</TR>" | "</OPTION>" | "<DD>" | "<DT>" | "<DL>" | "</UL>" | "</H1>" | "</H2>" | "</H3>" | "</H4>" | "</H5>" | "</H6>" | "</TH>"	Fin de línea
TABULADOR	"</TD>"	Tabulador
ENLACE	"<A" [^>]* ">"	Enlace HTML
FINENLACE	</A>	Fin de Enlace HTML
RADIO	"<INPUT" [^>]* TYPE=" [\"]? "RADIO" [^>]* ">"	Campo tipo Radio
OPTION	"<OPTION [^>]* ">"	Campo tipo Select
ENDOPTION	</OPTION>	Fin de opción Select
CHECKBOX	"<INPUT" [^>]* "TYPE=" [\"]? "CHECKBOX" [^>]* ">"	Campo checkbox
FINTABLA	"</TABLE>"	Marca fin de tabla

Figura III.4.3.3: Algunas marcas de formato ya incluidas en DEXTL

En cualquier caso, DEXTL permite definir fácilmente nuevas marcas de formato. Esto es útil, por ejemplo, cuando se desea extraer información de algún nuevo lenguaje de marcas. Por ejemplo DEXTL, además de para HTML, también incluye juegos de marcas para extraer datos desde el formato XTG de QuarkXPress[Qua02], y desde PDF [PDF00].

Tal y como se comentó previamente, una *marca de formato* viene definida por una expresión regular en el formato utilizado por los generadores de analizadores léxicos de la familia Lex[LEX75], tales como Jflex[JFL01]. Cada marca de formato lleva asociado un nombre. El formato para las definiciones de marcas de formato es: `NOMBRE=Expresión`, donde `NOMBRE` es el nombre de la marca y `Expresión` es la expresión regular.

Ejemplo III.4.3.12: Definición de marcas de formato (I)

Para HTML podemos definir la marca `FINLINEA` vista anteriormente de la siguiente manera: `FINLINEA= ("<br>" | "</p>"| "</tr>")`
□

A la vez que se detectan las marcas de formato, es posible asignar parte del texto que concordó con la expresión regular a identificadores que posteriormente pueden ser usados en los patrones de especificación, de manera similar a como se asignan a los atributos de una relación porciones del patrón. Esto se hace insertando en la expresión regular un identificador prefijado con el símbolo "$" en la posición dentro de la expresión regular del texto que se quiere asignar al identificador.

Ejemplo III.4.3.8: Definición de marcas de formato (II)

Supongamos que queremos definir una marca de formato `ENLACE` que concuerde con la marca `<A ...>` del lenguaje HTML. Supongamos además que queremos obtener el valor del atributo `href` contenido en dicha marcha HTML, asignándoselo a un identificador llamado URL. Podríamos escribir lo siguiente:

```
ENLACE="<a" [\r\n\t ]+ "href=" $URL " [^>]* >"
```

De esta manera, cuando se encuentre alguna porción de texto en el documento que concuerde con la expresión regular, además de identificar la marca de formato, se asignará al identificador URL la porción del texto que se encuentre entre `href=` y el siguiente espacio en blanco. Posteriormente a dicho espacio en blanco, puede ir opcionalmente cualquier combinación de caracteres diferentes de '>' (esto es lo que quiere decir la porción `'[^>]*'`) y, finalmente, el carácter '>', que cierra la marca HTML.

□

El valor de los identificadores obtenidos de esta manera puede ser utilizado en los patrones, por ejemplo para asignar el valor obtenido a alguno de los campos de las tuplas extraídas. La notación utilizada para esto es la siguiente:

```
NOMBREMARCAFORMATO (ATRIB1=ID1,..., ATRIBN=IDN)
```

dónde `ATRIB1,..., ATRIBN` son nombres de atributos en la relación cuyas tuplas se están extrayendo, e `ID1,...,IDN` son identificadores presentes en la definición de la marca de formato. De los atributos como `ATRIB1, ATRIB2...ATRIBN` diremos que están *ligados* a la marca `NOMBREMARCAFORMATO`.

Ejemplo III.4.3.9: Definición de marcas de formato (III)

Considérese una tienda electrónica de libros representada mediante la relación `R={TITULO,AUTOR,PRECIO,ENLACEAMASINFO}`, donde el campo `ENLACEAMASINFO` contiene una URL que permite acceder directamente a la página HTML de la tienda donde se proporciona información detallada sobre el producto. Si el aspecto gráfico de la salida es similar al mostrado en la Figura III.4.3.1 y suponemos definida la marca `ENLACE` de la misma manera que en el ejemplo anterior, además de la marca `FINLINEA`, definida de la manera habitual, podríamos escribir el siguiente patrón para extraer las tuplas de `R`.

```
PATRON R
MARCAS = {ENLACE, FINLINEA}
{
 ENLACE (ENLACEAMASINFO=URL) :TITULO FINLINEA
 NOINTERESA FINLINEA
 :AUTOR "/" NOINTERESA FINLINEA
 "Our Price:" PRECIO ¿",," NOINTERESA? FINLINEA
}
```

□

III.4.3.3.1.3. Separadores de Texto Avanzados

DEXTL incluye dos tipos de separadores de texto adicionales a los ya comentados: los separadores voraces y los separadores de subacadena.

III.4.3.3.1.3.1. Separadores voraces

Los separadores del tipo cadena de caracteres funcionan tratando de encajar con la primera ocurrencia del separador dentro del token de tipo TEXTO correspondiente.

Sin embargo a veces es útil, que los separadores traten de encajar con la última ocurrencia del separador dentro del token. Para ello basta prefijar el separador con el carácter ':'. A este tipo de separadores se les llama "voraces".

Ejemplo III.4.3.10: Separadores voraces

Supongase una tienda electrónica de libros que represente los diferentes autores de un libro separados por comas y que utilice este mismo separador para separar la lista de autores del precio del libro:

```
... Ron Rivest, Rick Adleman, 2000 ptas; ...
```

Si escribiésemos la especificación:

```
... :AUTORES "," :PRECIO ";" ...
```

entonces se asignaría al atributo AUTORES el valor 'Ron Rivest' y a PRECIO el valor 'Rick Adleman, 2000 ptas', lo cual probablemente no es el resultado deseado.

Sin embargo escribiendo:

```
... :AUTORES :"," :PRECIO ";" ...
```

se asignaría a AUTORES el valor 'Ron Rivest, Rick Adleman' y a PRECIO el valor '2000 ptas'.

□

III.4.3.3.1.3.2. Separadores de tipo subcadena

DEXTL permite dividir un token de texto en subelementos haciendo uso de índices posicionales en cadenas de caracteres.

Ejemplo III.4.3.11: Supongamos un token de texto representando un número telefónico. Supongamos que queremos dividir dicho token en dos segmentos: uno conteniendo los tres primeros caracteres del token, al que llamaremos PREFIJO y un

segundo segmento, al que llamaremos NUMERO, conteniendo los seis siguientes. Podríamos realizar esta tarea mediante la siguiente expresión en DEXTL.

```
... [(0-3:PREFIJO) (4-9:NUMERO)] ...
□
```

Como puede verse el token de texto se representa dentro del patrón encerrado entre los caracteres [y]. Cada segmento se encierra entre paréntesis, indicando la primera y la última posición de la cadena a asignar en el segmento. Si no se indicase última posición, se asumiría el final de la cadena como marca de fin para el segmento. Nótese que nada impide que los segmentos se superpongan entre sí.

III.4.3.3.1.4. Contextos de Navegación

A lo largo del proceso de extracción de tuplas realizado por un analizador, a menudo es conveniente disponer de un repositorio donde el analizador pueda almacenar valores que pueden serle útiles en etapas posteriores del análisis.

El generador de analizadores permite realizar esto a través del ya mencionado *contexto compartido* (ver sección III.4.1).

Como ya se ha dicho, el contexto consiste básicamente en una lista de pares campos-valor. Cada concordancia con un patrón en alguno de los documentos recorridos produce que se añada una entrada al contexto para cada atributo de la tupla que se ha "reconocido", tomando el nombre del atributo y el valor que le ha sido asignado en la tupla extraída. Si se extrajese más de una tupla de la relación durante la navegación por la secuencia (el caso más habitual), los pares campo-valor disponibles en el contexto serán los correspondientes a la última tupla extraída. Además cada relación tendrá accesible en el contexto la última tupla encontrada de todas sus relaciones padre.

Una vez introducidos en él, los valores del contexto pueden ser accedidos tanto desde las especificaciones de extracción como desde las especificaciones de NSEQL (como ya se ha visto en la sección III.4.2).

En las especificaciones DEXTL pueden utilizarse de las siguientes maneras:

- Como un separador del tipo cadena de caracteres. Esto permite que dichos separadores puedan ser obtenidos dinámicamente en una especificación.

Ejemplo III.4.3.12: Separadores de texto dinámicos usando variables del contexto

Considérese una fuente web que contiene tuplas para la relación:

```
PAIS (NOMBRE, NUMERO_HABITANTES, EXTENSION_EN_KMS)
```

Supongamos que para una aplicación determinada, se desea utilizar esta fuente de manera que consultándola por el nombre del país, devuelva los datos del mismo. Sin embargo, la interfaz para obtener los datos no consiste en un formulario que permita al usuario introducir el nombre del país y devuelva la información sobre el mismo, sino que se basa en una página que lista todos los países con un enlace a cada uno de ellos. Pulsando en el enlace correspondiente al país deseado, se accede a una página de detalle donde se encuentran los datos deseados.

El problema de construir un analizador que dado el nombre del país (supongámoslo introducido en el contexto con el nombre `PAIS`), obtenga la URL de la página que contiene sus datos, podría tratarse con la siguiente especificación:

```
ENLACE  (URLPAGINAPAIS=URL)  @PAIS FINENLACE
```

La URL a la página conteniendo los datos del país, quedaría almacenada en el contexto con el nombre `URLPAGINAPAIS`. En el apartado III.4.4 se verá como se podría usar dicha URL para acceder a la página y extraer los datos deseados.

□

- Dentro de la *invocación de acciones*, que se trata en al apartado III.4.3.3.1.5.

III.4.3.3.1.5. Acciones

DEXTL permite asignar *acciones* a un patrón. Las acciones se utilizan para ejecutar una operación arbitraria que es implementada por un código específico de la acción, y que es conforme a una determinada interfaz JAVA. Las acciones se ejecutarán una vez por cada concordancia encontrada con el patrón.

Si la *acción* se define *después del patrón principal* o *en medio* del mismo, la acción se ejecutará justo después de encontrar cada concordancia. Si se define *antes del patrón principal*, se ejecutará siempre antes de empezar a buscar una nueva ocurrencia.

Como consecuencia de esto, una acción definida *antes del patrón* se ejecuta siempre al menos una vez al empezar a buscar el patrón, incluso aunque no se encuentre ninguna tupla que concuerde con el mismo, mientras que una definida *después del patrón* sólo se ejecutará una vez encontrada alguna ocurrencia.

Las acciones pueden recibir un conjunto de pares campo-valor para parametrizar su ejecución. El valor de un parámetro de una acción puede ser:

- Constantes. No es necesario ponerlas entre comillas.
- Variables del contexto. Siguiendo el formato `@NOMBRE` (o `@{NOMBRE}` en caso de ser necesario delimitar hasta donde llega el nombre). Se sustituirán por su valor en el contexto en el momento de evaluarse la acción.

- Funciones. Siguiendo el formato: `^NOMBRE (param1,...,paramn)`. Se implementan conforme a una interfaz JAVA. Los parámetros, que pueden a su vez ser constantes, variables o, a su vez, otras funciones, se evalúan en función del contexto y después a partir de esos valores se evalúa la función.

Tras su ejecución, las acciones devuelven un valor booleano. Si el valor devuelto es `falso`, la tupla asociada a dicha ejecución de la acción es descartada y no se considera un resultado válido. Esto puede usarse para implementar restricciones de integridad sobre las tuplas de una relación.

La sintaxis para incluir una acción es la siguiente:

```
$NOMBREACCION (PARAM1=VALOR1,...PARAMi=@NOMBRE, PARAMn=VALORn)
```

dónde `NOMBREACCION` es el nombre de la acción (que sirve para identificar el componente de código que implementa la acción), `PARAM1,...,PARAMn` son los nombres de los parámetros que espera la acción. Para indicar que el valor de un determinado parámetro debe recogerse del contexto, se usa la notación `@NOMBRE` donde `NOMBRE` es la etiqueta que identifica en el *contexto de navegación* actual el valor deseado.

Si bien Wargo incluye las operaciones más frecuentes ya implementadas como *acciones*, el programador puede añadir nuevas *acciones* que reciban parámetros definidos por él y que realicen operaciones '*ad-hoc*'. Para ello, debe proporcionar una clase JAVA que ejecute la operación deseada y que se ajuste a la interfaz especificada.

Dos de las *acciones* que el sistema proporciona ya implementada son `IRRUTA` y `ANNADIRRUTA`, que son la base para combinar programas DEXTL con secuencias de navegación escritas en NSEQL. Este aspecto es tratado con detalle en el apartado III.4.4.

III.4.3.3.2. CONSTRUCCIÓN DE PROGRAMAS DEXTL

Una vez expuestos en detalle la estructura y la manera de interpretar los patrones DEXTL, es necesario describir algunos aspectos adicionales que son necesarios para la construcción de programas DEXTL completos.

En primer lugar, en la sección III.4.3.3.2.1, se describirá la manera de acotar el espacio de búsqueda de un patrón. Esto puede ser útil en varias circunstancias, pero especialmente para evitar situaciones de ambigüedad en las que el sistema podría extraer más concordancias con el patrón de las deseadas.

Posteriormente, el apartado III.4.3.3.2.2 describe el mecanismo de *patrones alternativos*, que permite tratar situaciones en las que el formato de los datos a extraer puede variar entre diversas consultas o seguir varios patrones distintos.

Finalmente, el apartado III.4.3.3.2.3 describe las construcciones necesarias para tratar con subrelaciones.

III.4.3.3.2.1. Delimitación de Regiones de Búsqueda

Dado que el analizador generado para extraer tuplas de una determinada relación funciona según un enfoque consistente en buscar las ocurrencias de un determinado patrón a lo largo de todo el documento, existe el problema potencial de que el patrón definido sea ambiguo. Lo que esto quiere decir es que es posible que haya porciones del documento que concuerden con el patrón pero que no se correspondan realmente tuplas de la relación.

La mayor parte de las veces esta situación puede evitarse alargando la especificación del patrón a reconocer con partes que, aunque no se deseen extraer, eviten la ambigüedad. Sin embargo, en otras ocasiones, esto no es tan sencillo. Un ejemplo típico se muestra en el siguiente ejemplo.

Ejemplo III.4.3.13: Ambigüedad de patrones

La figura III.4.3.4 muestra el formato gráfico utilizado por una tienda de libros en Internet para mostrar la información de sus productos.

Título	Autor	Precio
Corazón tan Blanco	Marías, Javier	2200 ptas.
Mañana en la batalla piensa en mí	Marías, Javier	2350 ptas.
Negra Espalda del Tiempo	Marías, Javier	3100 ptas.

Figura III.4.3.4: Resultados tabulados de una tienda de libros

Como puede verse, la información de los productos aparece tabulada. Esto tiene el inconveniente de que la fila cabecera de la tabla puede no ser fácil de distinguir de los resultados reales si sólo hemos definido la *marca de formato* que es más intuitiva en este caso: `TABULADOR="</td>"`.

De hecho una especificación como la siguiente:

```
PATRON LIBRO
MARCAS = {TABULADOR}
{
 TITULO TABULADOR AUTOR TABULADOR PRECIO TABULADOR
}
```

es ambigua, ya que devolvería la fila cabecera como si fuese una tupla válida de la relación 'Libro'.

□

Si bien en prácticamente todos los casos y también en este en particular, una cuidadosa definición del patrón utilizando *marcas de formato* adecuadas o que haga uso de las funcionalidades de tratamiento de subrelaciones (ver apartado III.4.3.3.2.3), permite definir un patrón no ambiguo, es evidente que sería más sencillo, rápido e intuitivo poder definir una región de búsqueda que englobe sólo a las filas de la tabla relevantes.

Para solucionar problemas como el planteado en el ejemplo anterior, el sistema ofrece la posibilidad de delimitar la parte de un documento dónde se desea buscar concordancias con un determinado patrón.

Dentro de un determinado nivel de la especificación, esto puede hacerse con las construcciones `DESDE-FINDESDE` y *marca de fin*.

La construcción `DESDE-FINDESDE` se utiliza para delimitar *por arriba* (esto es desde el principio del documento hacia el final del mismo) la zona de búsqueda de concordancias. El formato de esta construcción se muestra a continuación:

```
{
 ...
 DESDE
  Patrón Desde
 FIN_DESDE
 Patrón Principal Del Nivel
 ...
}
```

Dónde `Patrón Desde` es un patrón construido de la misma manera que los patrones habituales, y del cual, cuando se encuentra la primera ocurrencia, se abandona la búsqueda del mismo, pasándose a buscar ocurrencias del patrón principal del nivel. Si no se encuentra ninguna ocurrencia de `Patrón Desde`, el conjunto de tuplas extraído para el nivel será vacío.

Como puede verse, la construcción `DESDE-FINDESDE` debe utilizarse dentro de la especificación de un nivel, antes de la definición del patrón principal del nivel.

En cuanto a la *marca de fin*, su función es simétrica pero delimitando *por abajo* la zona de búsqueda de concordancias. El patrón *marca de fin* debe insertarse entre los símbolos '<' y '>' inmediatamente antes de la marca de fin del subpatrón '}'. Si no se incluye marca de fin, se asume implícitamente que ésta es el fin del documento actual.

De esta manera, para el ejemplo anterior, la siguiente especificación soluciona el problema de ambigüedad:

```
PATRON LIBRO
MARCAS = {TABULADOR}
{
 DESDE
 "Titulo" TABULADOR "Autor" TABULADOR "Precio" TABULADOR
 FIN_DESDE
 TITULO TABULADOR AUTOR TABULADOR PRECIO TABULADOR
}
```

Si la búsqueda de concordancias se desea restringir a más de una zona del documento, puede conseguirse fácilmente repitiendo la especificación del nivel tantas veces como zonas haya y utilizando `DESDE-FINDESDE` y la *marca de fin de nivel* en cada una de las especificaciones para delimitar las zonas.

Ejemplo III.4.3.14: Ambigüedad de patrones (II)

La siguiente especificación busca ocurrencias del patrón `Patrón` en las dos zonas del documento delimitadas de la siguiente manera:

La primera zona abarca desde la primera ocurrencia en el documento de `PatrónDesde1` hasta la primera ocurrencia en el documento de `PatrónHasta1` posterior a `PatrónDesde1`.

La segunda zona abarca desde la primera ocurrencia en el documento de `PatrónDesde2` posterior a la primera ocurrencia de `PatrónHasta1`, hasta la primera ocurrencia en el documento de `PatrónHasta2` posterior a `PatrónDesde2`.

```
{
 ...
 DESDE PatrónDesde1 FIN_DESDE
 Patrón
 <PatrónHasta1>
}
{
 ...
 DESDE PatrónDesde2 FIN_DESDE
 Patrón
 <PatrónHasta2>
}
```

□

III.4.3.3.2.2. PATRONES ALTERNATIVOS

Hasta ahora hemos considerado que para extraer las tuplas de una determinada relación podía utilizarse un único patrón. Sin embargo, es posible especificar varios patrones alternativos de manera que se genere un nueva tupla de la relación por cada concordancia de cualquiera de los patrones.

Esto es útil en varias situaciones. En primer lugar, es una alternativa al uso de porciones opcionales o alternativas en el interior de los patrones.

También es útil cuándo el formato en que están representadas las tuplas de una determinada relación puede variar en función de la consulta realizada sobre la fuente, pero sin que sea posible determinar *a priori* cuál de los formatos de salida va a ser utilizado.

Ejemplo III.4.3.15: Patrones alternativos

En muchas tiendas electrónicas, cuando una búsqueda ofrece un solo producto como resultado, se salta directamente al formato de '*página de detalle del producto*' en lugar de a la habitual página de '*resultados de búsqueda*'. Sin embargo, antes de realizar la búsqueda no es posible saber si ésta va a devolver como resultado uno o varios productos, con lo que no es posible determinar *a priori* que patrón debe utilizarse para extraer las tuplas de la relación.

□

DEXTL permite utilizar el símbolo '||' para separar los diferentes *patrones alternativos* que extraen datos para una misma relación. El analizador buscará las ocurrencias de todos los patrones y creará nuevas tuplas para las ocurrencias de todos ellos.

Así, para el ejemplo anterior, podría construirse una especificación como la siguiente:

```
{
 {
   //Patrón de la página de resultado de búsqueda
  .....
 }
 ||
 {
   //Patrón de la página de detalle de producto
  .....
 }
}
```

III.4.3.3.2.3. Niveles, Relaciones y Subrelaciones

Tal y como se comentó en el apartado III.4.3 de este documento, un atributo de una determinada relación puede ser a su vez una relación con sus propios atributos. El generador de analizadores debe por lo tanto proporcionar soporte para tratar estos casos.

Cuando esto ocurra se dirá que la subrelación asociada al atributo "es hija" de la relación que contiene el atributo.

Lo primero que es necesario recalcar es que un atributo del tipo sub-relación no presenta ninguna diferencia en lo que se refiere al cumplimiento de las heurísticas presentadas en el apartado III.4.3.2. Considerado en su totalidad como un atributo que contiene un conjunto de tuplas en lugar de un valor atómico, debe cumplir igualmente las heurísticas de *separación vertical* y *unidad de tupla*. Por otra parte, considerado como relación, se asume también que cumple todas las heurísticas por las mismas razones por las que lo hace una relación normal.

Para que el sistema sea capaz de extraer tuplas de una subrelación es necesario definir un nuevo nivel para ellas. Cada nivel tiene un nombre, que debe coincidir con el de la relación o subrelación cuyas tuplas extrae y que se indica con la notación `PATRON NOMBREPATRON`, donde `NOMBREPATRON` es el nombre asignado al nivel. La manera de expresar la especificación del subnivel es incluirla entre los símbolos '{' y '}'. Se dirá que el patrón que trata de extraer tuplas de la subrelación pertenece a un *subnivel* con respecto a aquel que busca tuplas de la relación padre.

Además, para que el sistema sea capaz de saber cuando debe dejar de buscar tuplas de la subrelación y volver a buscar tuplas de la relación de nivel superior, es necesario indicar un patrón que delimite el final de la subrelación. Este patrón constituye una *marca de fin de la subrelación* y debe insertarse entre los símbolos '<' y '>' inmediatamente antes de la marca de fin del subpatrón '}'. Que esta *marca de fin* existe está garantizado por la heurística de *separación vertical*. Si no se incluye marca de fin, se asume implícitamente que ésta es el fin del documento actual. Un detalle importante es que el texto que causa que se encuentre la *marca de fin*, es devuelto a la entrada una vez que se ha abandonado el subnivel. De esta manera, dicho texto será considerado de nuevo en el nivel padre.

Ejemplo III.4.3.16: Subniveles

Consideremos una tienda electrónica de películas representada por la relación:

```
R={TITULO, ACTORES, DIRECTOR, GENERO, RESUMEN, EDICION{DESCRIPCION,
FORMATO, PRECIO}}
```

La figura III.4.3.5 muestra un ejemplo en el que se observa un fragmento de una página web en la que se representa gráficamente una tupla de `R`. El formato de

resultado es similar al utilizado por la tienda electrónica Reel [Ree], si bien ha sido modificado por razones de brevedad.

Small Time Crooks (2000)
Starring: Woody Allen, Tracey Ullman
Director: Woody Allen
Synopsis: Manic crime comedy about an affable ex-con (Allen) cooking up a cunning bank robbery while working in a restaurant.
Runtime: 95 minutes
MPAA Rating: PG
Genre: Comedy

Description:	Format:	List:	Price:	Availability:
Small Time Crooks (Widescreen)	DVD	$26.99	$20.99	Pre-Order
Small Time Crooks	VHS	$14.99	$10.49	Pre-Order
Small Time Crooks (Spanish Subtitles)	VHS	$14.99	$10.49	Pre-Order

Figura III.4.3.5: Formato de Salida de una tienda electrónica de películas

Para construir esta especificación, el primer patrón utilizará sólo la marca de formato `FINLINEA`, mientras el segundo utilizará tres: `FINLINEA`, `TABULADOR`, y `FINTABLA`. `FINTABLA` representa el fin de una tabla en HTML. Así, `FINTABLA="</table>"`.

La especificación que sería capaz de extraer tuplas con este formato sería la siguiente:

```
PATRON R
{
 MARCAS = {FINLINEA}
 TITULO " (" NOINTERESA ")" FINLINEA
 "Starring:" ACTORES FINLINEA
 "Director:" DIRECTOR FINLINEA
 "Synopsis:" RESUMEN FINLINEA
 NOINTERESA FINLINEA
 NOINTERESA FINLINEA
 "Genre:" GENERO FINLINEA
 PATRON EDICION
 {
  MARCAS = {TABULADOR, FINLINEA, FINTABLA}
   DESDE "Description:" TABULADOR "Format:"TABULADOR "List:"
TABULADOR "Price:" TABULADOR "Availability:" TABULADOR) FINDESDE

   :DESCRIPCION TABULADOR :FORMATO TABULADOR NOINTERESA TABULADOR
:PRECIO TABULADOR
  <FINTABLA>
 }
```

El código HTML asociado a la figura III.4.3.5 se muestra en la figura III.4.3.6 (habiendo suprimido ya todas aquellas marcas o "tags" HTML que, al no concordar con ninguna de las definiciones de *marcas de formato*, son ignoradas por el analizador léxico).

```
Small Time Crooks  (2000)<BR>
Starring: Woody Allen, Tracey Ullman<BR>
Director: Woody Allen<BR>
Synopsis: Manic crime comedy about an affable ex-con  (Allen) cooking
up a cunning bank robbery while working in a restaurant.<BR>
Runtime: 95 minutes<BR>
MPAA Rating: PG<BR>
Genre: Comedy<BR>

</TD>
Description:</TD>Format:</TD>List:</TD>Price:</TD>Availability:</TD>
 Small Time Crooks  (Widescreen)</TD>DVD</TD>$26.99</TD>
$20.99</TD>Pre-Order</TD>
</TR>
< Small Time Crooks</TD>VHS</TD>$14.99</TD>$10.49</TD>Pre-Order</TD>
</TR>
< Small Time Crooks  (Spanish
Subtitles)</TD>VHS</TD>$14.99</TD>$10.49</TD>Pre-Order</TD>
</TR>
</TABLE>
```

Figura III.4.3.6: Código HTML "relevante" para la Figura III.4.3.5

El analizador sintáctico funcionará ahora de la siguiente manera:

- Hasta llegar al comienzo de la subrelación funciona de la misma manera que se ha visto hasta el momento.
- Cuando, para una tupla, llega al momento en que tiene que empezar a buscar tuplas de la subrelación:
- Realiza el cambio de juego de marcas si es distinto del nivel padre (en este caso es así).
- Comienza a buscar simultáneamente el patrón de las tuplas de la subrelación y el patrón de *marca de fin de la subrelación*.
- La búsqueda de tuplas de la subrelación se desarrolla de la misma manera que la búsqueda de tuplas en un nivel padre.
- Cuando se encuentra la *marca de fin de la subrelación*, el conjunto de tuplas de la subrelación extraídas pasan a ser el valor del atributo correspondiente de la tupla del nivel padre. Además se devuelve a la entrada todo el texto que se haya considerado desde que se encontró la última tupla de la subrelación, incluído el que desencadenó el reconocimiento de la marca de fin.
- Al finalizar el tratamiento del subnivel, se prosigue buscando los atributos que falten de la tupla del nivel padre.

□

III.4.3.3.2.3.1. Niveles de Acción

Además de los niveles que pretenden extraer tuplas de una relación o subrelación, existen otro tipo de niveles soportados por el sistema. Su funcionamiento es idéntico al de los niveles anteriores, con la diferencia de que las ocurrencias del patrón buscado se añaden al contexto, pero no se devuelven a la aplicación llamante. Son, en definitiva, niveles para recolectar información intermedia que puede ser útil durante el proceso de extracción de tuplas, pero que no son tuplas en sí.

Estos niveles son especialmente útiles cuando se combinan con *acciones*. Las *acciones* fueron tratadas en el apartado III.4.3.3.1.5 de este documento.

Los niveles de acción se distinguen de las anteriores en la notación utilizada para la apertura del nivel, que se efectúa con los símbolos '{-' y '-}'.

III.4.3.4. EJEMPLOS COMPARATIVOS DE USO

A pesar, de que como ya se ha comentado y como se detallará en la sección III.4.5, el usuario de Wargo no precisa utilizar directamente DEXTL para la construcción de programas de extracción, sino que puede utilizar una herramienta gráfica más sencilla, es conveniente destacar que, incluso sin dicha herramienta, DEXTL es significativamente más simple de utilizar que los lenguajes de intenciones similares presentados previamente.

Para ilustrar esto, incluimos aquí las especificaciones precisas en nuestro sistema para tratar las mismas fuentes que se trataron como ejemplo para los sistemas [HGC97] (ejemplo II.5.4.4) [GRV98](ejemplo II.5.4.5) y [KM98] (ejemplo II.5.4.6).

Ejemplo III.4.3.17: Especificaciones para el ejemplo II.5.4.4.

El siguiente patrón permite extraer la información deseada en nuestro sistema.

```
PATRON PRONOSTICO
MARCAS = {TABULADOR, FINLINEA}
 {
   DESDE
   "country" TABULADOR "city" TABULADOR "forecast"  TABULADOR
"hi/lo:" TABULADOR "forecast" TABULADOR "hi/lo")
   FINDESDE

   :PAIS TABULADOR :CIUDAD TABULADOR :PRONOSTICOHOY TABULADOR
:TEMPHOY TABULADOR :PRONOSTICOMANNANA TABULADOR :TEMPMANNANA FINLINEA
 }
```

Como puede verse, la original era bastante más compleja:

```
1 [["root",
 2   "get ('http://www.intellicast.com/weather/europe/')",
```

```
 3    "#"
 4   ],
 5   ["temperatures",
 6    "root",
 7    "*<TABLE*<TABLE*</TR>#</TABLE>*"
 8   ],
 9   ["_citytemp",
10    "split (temperatures,'<TR ALIGN=left>')",
11    "#"
12   ]
13   ["city_temp"
14    "_citytemp[1:0]",
15    "#"
16   ]
17  ["country,c_url,city,weath_tody,hgh_tody,low_today,weath_tomorrow,
       hgh_tomorrow,low_tomorrow",
18    "city_temp",
19
"*<TD>#</TD>*HREF=#>#</A>*<TD>#</TD>*<TD>#/#</TD>*<TD>#</TD>*<TD>#/#*
"
20   ]]
```

[HGC97] precisa de especificaciones muy complicadas incluso para casos sencillos como el mostrado en el ejemplo (una salida de datos tabulados). Además, la especificación a escribir es muy dependiente de parámetros internos al código HTML de la página, mientras que nuestra especificación "reproduce" el aspecto visual de la página, por lo que puede construirse de manera más intuitiva (en la mayoría de los casos, incluso sin ver el código fuente de la pagina)

□

Ejemplo III.4.3.18: Especificaciones para el ejemplo II.5.4.5.

En este ejemplo no se proporciona información exacta sobre la tabla completa de la que se extraen los datos. Sólo se nos dice que los datos a extraer están en la segunda columna. Una especificación de ese tipo se construye trivialmente en nuestro sistema, sea cual sea el contenido de las otras dos columnas. Aquí supondremos sólo dos. La especificación en nuestro sistema sería de la forma siguiente:

```
PATRON PROVINCIA
MARCAS = {TABULADOR, FINLINEA}
 {
   DESDE
   //Patron cabecera de la tabla
   "cabecera1" TABULADOR "Provincia" FINLINEA
   FINDESDE

   NOINTERESA TABULADOR :PROVINCIA FINLINEA
 }
```

frente a la de [GRV98], que era:

```
1     ProvinciasGalicia: Root.child[Name=table &
Title="Galicia"].child.child[Name=td & Occurrence=2].Data
```

Frente a [HGC97], [GRV98] proporciona un lenguaje más compacto y la posibilidad de generar gráficamente parte de la especificación. Pero, especialmente para fuentes que sean más complejas que la de este sencillísimo ejemplo, la dificultad de uso del lenguaje seguirá siendo elevada. Para fuentes como la mostrada en la Figura III.4.3.1 o más complejas, la complejidad sería también muy grande.

□

Ejemplo III.4.3.19: Especificaciones para el ejemplo II.5.4.5.

El siguiente patrón permite extraer la información deseada en nuestro sistema (al igual que en la especificación de WebL, recuperamos sólo el símbolo y la última cotización).

```
PATRON COTIZACION
MARCAS ={TABULADOR, FINLINEA}
 {
   DESDE
   "Stock Quotes"
   FINDESDE

   :SIMBOLO TABULADOR NOINTERESA TABULADOR NOINTERESA TABULADOR
NOINTERESA TABULADOR NOINTERESA TABULADOR :COTIZACION TABULADOR
 }
```

La especificación original (incluimos sólo la extracción de los datos en sí) era:

```
pagina.Elem ("B") in  (pagina.Elem ("TABLE") contain pagina. Pat
("Stock Quotes"))[0])[1].Text ()
4) end;
```

Nuevamente, este sistema precisa de especificaciones bastante complicadas incluso para casos tan sencillos como el de una salida de datos tabulados. Nuestra especificación "reproduce" el aspecto de la página en el navegador, por lo que puede construirse de manera más intuitiva (en la mayoría de casos, incluso sin ver el código fuente de la pagina).

□

III.4.4. NAVEGACIÓN ENTRE DOCUMENTOS: COMBINANDO NSEQL Y DEXTL

Tal y como se expuso en la sección III.4.1.1, en tiempo de ejecución Wargo recibe una lista de rutas que son recorridas secuencialmente para ejecutar programas de extracción escritos en DEXTL (cada ruta puede tener asociado un programa distinto).

Sin embargo, con frecuencia ocurre que las rutas a recorrer no son conocidas en tiempo de generación del envoltorio, sino que su número exacto y el programa NSEQL

requerido no pueden determinarse hasta el tiempo de ejecución. Considérese el siguiente ejemplo para ilustrar este punto.

Ejemplo III.4.4.1: Navegación para completar una tupla

Consideremos una tienda electrónica de libros modelada como una relación R={TITULO, AUTOR, EDICION{EDITORIAL, PRECIO, FORMATO}}. En esta tienda, las páginas de resultado de búsqueda tienen el siguiente formato:

```
Jurassic Park. Michael Crichton. 20 €.
```

Como puede verse, la página de resultado de la búsqueda sólo ofrece como información de cada tupla los campos TITULO, AUTOR y PRECIO. Sin embargo, el enlace asociado al TITULO del libro permite acceder a una página de información detallada donde se proporciona una lista de EDICIONES del libro. Por lo tanto, el envoltorio podría atravesar dicho enlace y extraer los valores requeridos para completar el atributo.

Sin embargo, no hay manera de que el creador del envoltorio pueda saber a priori cuantas rutas serán necesarias (dependerá del número de resultados de la consulta) ni cuáles serán exactamente los enlaces que deben ser atravesados (ya que estos enlaces se generan también dinámicamente como respuesta a la consulta efectuada).

Otro ejemplo común surge con los resultados multi-paginados. Un ejemplo de esto ocurriría si esta tienda dividiese los resultados a una consulta dada en varias páginas, mostrando sólo un cierto número de resultados por página y permitiendo el acceso al resto de páginas mediante enlaces Anterior y Siguiente. Nuevamente, es imposible para el creador del envoltorio saber a priori cuantas páginas de resultados será necesario recorrer (y, por lo tanto, cuántos enlaces Siguiente habrá que atravesar), ya que este dato dependerá del número de resultados obtenidos para cada consulta efectuada.

□

Wargo permite atacar estos problemas haciendo uso de las *acciones* de los patrones DEXTL (ver sección III.4.3.3.1.5). En concreto, Wargo permite que estas acciones accedan a la lista de rutas para añadir o suprimir rutas a conveniencia.

Wargo proporciona dos acciones ya implementadas que permiten añadir rutas dinámicamente a la lista, y que son suficientes para todas las situaciones que hemos planteado hasta el momento. Su funcionamiento se describe en el siguiente sub-apartado.

III.4.4.1. IRRUTA/ANNADIRRUTA

IRRUTA y ANNADIRRUTA son acciones predefinidas que pueden ser utilizadas en los programas DEXTL para añadir dinámicamente rutas a la *secuencia de navegación* actual.

Ambas funciones permiten especificar rutas, tanto en formato http, como en formato NSEQL. En su versión para rutas http convencionales, IRRUTA debe recibir dos parámetros:

- MÉTODO. Los valores válidos son GET y POST para el caso de rutas HTTP. Indica el método que debe utilizarse en la petición HTTP.
- URLPATRÓN. Recibe una URLPatron con el formato expuesto en la sección III.4.2.

En la versión con rutas definidas en NSEQL, IRRUTA recibe un único parámetro con una secuencia de comandos separadas por el carácter '*;*'.

IRRUTA tiene como efecto añadir una nueva ruta en el punto actual de la secuencia de navegación, provocando un salto a la ruta indicada en la llamada.

Ejemplo III.4.4.2: IRRUTA

Consideremos de nuevo el ejemplo de una tienda electrónica de libros modelada como una relación:

```
LIBRO={TITULO, AUTOR, EDICION{EDITORIAL, PRECIO, FORMATO}}
```

En esta tienda, las páginas de resultado de búsqueda tienen el siguiente formato:

```
Jurassic Park. Michael Crichton. 20 €
```

La página de resultado de la búsqueda sólo ofrece como información de cada tupla los campos TITULO, AUTOR y PRECIO. El enlace asociado al TITULO del libro permite acceder a una página donde se proporcionan una lista de EDICIONES.

Una especificación posible para extraer las tuplas de R sería la siguiente:

```
PATRON R
MARCAS = {ENLACE, FINENLACE, FINLINEA}}
{
 ENLACE  (URLPAGINADETALLE=URL)  :TITULO FINENLACE
 $IRRUTA ("CLICKONANCHORBYHREF (@URLPAGINADETALLE)")
 {
  //Patrón que extrae EDITORIAL,FORMATO y DESCRIPCIÓN
  ...
 }
 :AUTOR "." PRECIO FINLINEA
}
```

Como puede verse, se extraen las URLs de los enlaces sobre el campo TITULO (que se almacenan en la variable del contexto URLPAGINADETALLE) para añadir dinámicamente rutas, que se construyen con la acción CLICKONANCHORBYHREF. Esta función recibe una cadena de caracteres y atraviesa el enlace cuyo valor para el atributo href coincide con el parámetro recibido. Entonces el sistema navega a dichas páginas y extrae las tuplas del atributo compuesto EDICION.

□

Además de IRRUTA, existe otra acción predefinida para propósitos de navegación entre documentos, que se llama ANNADIRRUTA. La única diferencia en su funcionamiento con IRRUTA es que la nueva ruta se ejecutará tras el término del procesamiento del documento actual.

Ejemplo III.4.4.3: ANNADIRRUTA

Un ejemplo típico del uso de ANNADIRRUTA es el de permitir al analizador extraer tuplas de más de un documento, en una lista de resultados multi-página.

Una manera de tratar estos casos en un programa DEXTL se muestra a continuación:

```
PATRON R
{
 //Tuplas a extraer
  ...
}
{-
  ENLACE (URLSIGINTERVAL=URL) "Intervalo Siguiente" FINENLACE
  $ANNADIRRUTA (METODO="GET", URLPATRON="@URLSIGINTERVAL")
}
```

En este caso, se ha utilizado una petición http (podría utilizarse sin problemas una ruta de tipo secuencia de acciones).

Si bien podría conseguirse un efecto muy parecido empleando $IRRUTA, usar $ANNADIRRUTA es aquí más natural ya que no se desea saltar al nuevo documento hasta haber terminado totalmente el actual.

□

III.4.5. HERRAMIENTAS GRÁFICAS DE GENERACIÓN DE ENVOLTORIOS

Con el fin de facilitar todavía más la creación de programas envoltorio para fuentes web, se ha realizado una herramienta gráfica que permite la construcción visual de las secuencias de navegación y de las especificaciones de extracción de información. Este epígrafe describe las ideas fundamentales en las que se basa el funcionamiento de dicha herramienta.

Esta consiste básicamente en un explorador comercial de Internet, que ha sido extendido con varios menús y botones específicos para nuestros propósitos. Las funciones accesibles desde estos, permiten la generación de programas DEXTL y NSEQL de manera gráfica.

III.4.5.1. GENERACIÓN DE PROGRAMAS DEXTL

El primer paso en la generación de programas DEXTL es identificar el modelo de salida de datos del envoltorio.

Para ello el usuario dispone de una interfaz que le permite definir una jerarquía de relaciones y sub-relaciones. Para cada relación, el usuario especificará su lista de atributos. Para los atributos atómicos, el usuario debe especificar únicamente su nombre y, opcionalmente, su tipo de dato. Para las sub-relaciones, se especificarán sus propios atributos.

Una vez que el esquema de salida ha sido creado, el usuario debe crear patrones DEXTL para cada una de las sub-relaciones. La herramienta impone un orden de creación de los patrones necesarios, dónde los patrones correspondientes a elementos de información padre deben crearse antes que los de sus hijos. Para cada sub-relación deberá además crearse un patrón DEXTL adicional para la *marca de fin*. El patrón de *marca de fin* de una sub-relación debe crearse antes que los patrones de extracción de datos. Los patrones `DESDE` pueden ser creados también en este punto.

Para crear un patrón DEXTL, se comienza seleccionando con el ratón una porción de la página que represente una tupla acorde al patrón a extraer.

Entonces el usuario puede utilizar la opción '`Generar Patrón`'. En primer lugar, el sistema interroga al usuario para obtener una serie de metainformación que se puede asignar a cada patrón, para que el sistema pueda mejorar la generación de los patrones de especificación. El siguiente recuadro muestra una especificación de metainformación válida, para un patrón `LIBRO` en una aplicación de compra comparativa.

```
MARCAS={FINLINEA}
SEPARADORES_TEXTO={"/", ";"}
EJEMPLOS
{
 {
   TITULO=@TITULO
  AUTORES=@AUTORES
 }
}
```

Esta metainformación consta del conjunto de marcas de formato a utilizar, de un conjunto de separadores de texto habituales en el dominio, que el sistema asumirá por defecto, y de una lista de ejemplos de tuplas válidas.

La asignación de los valores `@TITULO` y `@AUTORES` a los atributos `TITULO` y `AUTORES` significa que el sistema debería usar como ejemplos los valores proporcionados por el usuario para esos atributos al rellenar el último formulario HTML atravesado.

Entonces la herramienta obtiene el código HTML asociado a la porción de la página marcada por el usuario y se lo entrega al sistema analizador, que trocea el código recibido de acuerdo al juego de marcas de formato especificado para el patrón. De esta manera se obtiene un patrón preliminar compuesto por textos y marcas de formato. Entonces trata de dividir los tokens de texto de acuerdo a los separadores de tipo cadena de caracteres incluidos en la metainformación del patrón. Finalmente intenta asignar a cada porción de texto un atributo del patrón comparando su valor con el de los ejemplos proporcionados. Si no hay ningún ejemplo que coincida, se asigna un `NOINTERESA` a esa porción de texto.

Por ejemplo, si el usuario marcase el primer resultado de búsqueda de la tienda Barnes & Noble mostrado en la Figura III.4.3.1, el patrón generado sería:

```
PATRON LIBRO
MARCAS = {FINLINEA}
{
 :TITULO FINLINEA
  NOINTERESA FINLINEA
  :AUTORES "/" NOINTERESA "/" NOINTERESA "/" NOINTERESA FINLINEA
  NOINTERESA  FINLINEA
}
```

Para finalizar la especificación del patrón, el usuario debe aún: 1) Trocear los tokens de texto obtenidos, si es necesario, utilizando separadores de tipo cadena de caracteres 2) Asignar a cada token de texto, o bien un `NOINTERESA` o bien el nombre de un atributo del patrón. Para todas estas tareas, la herramienta incluye interfaces interactivas que guían paso a paso al usuario en el proceso.

Una vez que el patrón está generado, Wargo resaltará automáticamente en el navegador todas las ocurrencias en el documento del patrón definido. Si alguna ocurrencia que debería haber sido seleccionada no lo ha sido, el usuario puede sencillamente añadir un nuevo patrón para la subrelación, marcando con su ratón una de las ocurrencias no resaltadas y repitiendo el proceso de generación del patrón, lo cuál generará un patrón alternativo (ver sección III.4.3.3.2.2) para extraer tuplas de la subrelación.

Si Wargo selecciona más tuplas de las deseadas, el usuario puede refinar el patrón o delimitar la región de búsqueda de concordancias, utilizando las técnicas expuestas en el apartado III.4.3.3.2.1.

Cuando el usuario está satisfecho con los patrones generados, puede probarlos extrayendo las tuplas de la página escogida o de otras similares, utilizando para ello el módulo de tiempo de ejecución de Wargo.

Cuando el patrón esté ya probado, el usuario puede utilizarlo para obtener nuevos ejemplos. Así, 'Wrox Press' y 'Sybex Incorporated' podrían ser nuevos ejemplos para el atributo EDITORIAL.

III.4.5.2. GENERACIÓN DE PROGRAMAS NSEQL

Para generar una secuencia de navegación, un usuario de la herramienta puede utilizar la opción 'Comenzar secuencia' para hacer que el explorador cambie a 'modo grabación'. En este modo de funcionamiento, la herramienta es capaz de grabar las navegaciones realizadas por el usuario, de manera que posteriormente se pueden obtener de forma automática los comandos que implementan dicha secuencia.

Según el usuario navega, el sistema le pide información para completar las partes variables de los comandos de las secuencias. Por ejemplo, cuando el usuario atraviese un formulario en una página web, habrá establecido una serie de valores en los campos de dicho formulario. Por defecto, el sistema podría asumir que los valores proporcionados por el usuario al atravesar el formulario son los correctos para ese paso de la secuencia. Sin embargo, esta no tiene por qué ser la decisión correcta: quizás los valores a utilizar deben ser especificados en tiempo de ejecución por el mediador para contestar una consulta determinada.

Por ello, cuando el usuario atraviese un formulario, se lanzará una ventana emergente del navegador en la que aparecerá una reproducción del formulario que acaba de atravesar. Para cada campo de ese formulario, el sistema presentará al usuario una serie de opciones. Básicamente, el usuario podrá escoger o bien un valor constante o bien una variable que esté actualmente en el contexto de navegación. Si el campo admite sólo ciertos valores constantes (esto ocurrirá por ejemplo si es una lista de selección), el sistema presenta como posibles sólo los valores válidos.

En el ejemplo anterior, el usuario escogería la variable del contexto @TITULO para el campo título del formulario, y la variable @AUTOR para el campo autor. De esta manera, el sistema podrá generar fácilmente de forma automática la siguiente secuencia:

```
1) NAVIGATE ("http://sitioweb.com/");
2) CLICKONANCHORBYTEXT ("Buscar libros");
3) FINDFORMBYNAME ("Libros",0);
4) SETINPUTVALUE (campotitulo,0,"@TITULO");
5) SETINPUTVALUE (campoautor,0,"@AUTOR");
6) SUBMITFORM ();
```

Los dos primeros comandos serían detectados por la herramienta de manera automática al grabar los pasos efectuados por el usuario. Representan, respectivamente, la conexión a la web de la tienda electrónica y el acceso a la página de consulta atravesando un enlace cuyo texto sea "`Buscar libros`".

El tercer comando también es generado por la herramienta comprobando qué formulario ha sido el utilizado por el usuario. Sin embargo, para obtener adecuadamente los comandos 4) y 5), precisa asistencia del usuario, de la manera que ya hemos comentado, para obtener los valores para los dos campos del formulario.

Si el destino de la secuencia de navegación fuese una página con varios marcos (o `frames`), y se desease restringir la extracción a los contenidos de un marco concreto, bastaría con que el usuario resaltase con el ratón una parte cualquiera de los contenidos de ese marco, y entonces seleccionase la opción '`Seleccionar marco`'. De esta manera, el sistema añadiría automáticamente un comando '`SelectFrame`'.

Una vez que el usuario llega a una página de la que desea extraer información, puede cerrar la secuencia de navegación con la opción '`Finalizar Secuencia`' del menú.

III.4.5.2.1. RUTAS AÑADIDAS EN TIEMPO DE EJECUCIÓN

Como ya se ha comentado, un patrón puede llevar insertadas acciones del tipo `IRRUTA` para efectuar secuencias de navegación durante el proceso de extracción (por ejemplo, para acceder a otra página de la que seguir extrayendo datos).

El usuario debe indicar en que patrón debe insertarse la acción y qué acción de las disponibles (`IRRUTA` o `ANNADIRRUTA`) desea incluir. En ese momento, el usuario podrá generar una secuencia de navegación de una manera similar a la ya mostrada en el apartado anterior. El siguiente ejemplo ilustra el proceso.

Ejemplo III.4.5.1: Insertando visualmente secuencias de navegación con `IRRUTA`

Supóngase una web de banca electrónica. Cuando un usuario accede a su página de información sobre cuentas corrientes, se le presenta un formulario con un elemento de tipo `radiobutton` por cada una de sus cuentas corrientes. Para cada cuenta se

muestran dos columnas de información: una con el número de cuenta y otra con el saldo de la misma. El usuario puede seleccionar una de sus cuentas (marcando el elemento `radiobutton` asociado) y pulsar 'Aceptar', obteniendo así los movimientos para la cuenta seleccionada.

Para extraer los datos de cuentas corrientes del usuario puede construirse una especificación como esta:

```
PATRON CUENTACORRIENTE
MARCAS = {RADIO, TABULADOR, FINLINEA}
{
 RADIO(IDCUENTA=value) NUMCUENTA TABULADOR SALDO FINLINEA
}
```

donde se han definido las marcas `FINLINEA` y `TABULADOR` de la forma habitual, y se ha definido una marca `RADIOBUTTON` que concuerda con la representación en HTML de dichos elementos (que es del tipo `<input type="radio" name="nombre" value="valor">`).

Para extraer los movimientos de cada cuenta corriente, es necesario invocar el formulario asignando el valor extraído en `IDCUENTA` al campo tipo `RADIOBUTTON`. En la página obtenida, deben extraerse los movimientos. Es decir, la especificación completa será similar a la siguiente:

```
PATRON CUENTACORRIENTE
MARCAS = {RADIO, TABULADOR, FINLINEA}
{
 RADIO(IDCUENTA=value) NUMCUENTA TABULADOR SALDO FINLINEA
$IRRUTA('FindFormByName("NombreForm",0);SetInputValue(NombreCampoRadi
oButton, 0, @IDCUENTA); submitform();')

 PATRON MOVIMIENTO
 {
  // Patrón que extrae los movimientos de la cuenta
    ...
 }
}
```

Para generar la secuencia de navegación de la especificación anterior, el método a seguir sería sencillamente escoger la opción "`ComenzarSecuencia`", seleccionar una cuenta cualquiera del formulario e invocarlo para obtener los movimientos. La herramienta pedirá al usuario que complete la información sobre el formulario en la forma habitual. Este seleccionará el valor `@IDCUENTA` (que estará en el contexto, dada la posición dentro del patrón de la acción `IRRUTA`), para el campo del formulario. De esta manera, la herramienta puede generar la secuencia por sí misma e incluirla en la acción `IRRUTA`.

IV. EXPERIENCIA OBTENIDA DEL USO DEL SISTEMA: APLICACIONES Y EVALUACIÓN

Esta sección presenta una evaluación del comportamiento del sistema mediador presentado en este trabajo, basándose para ello en los datos recogidos de su aplicación para diversos proyectos en entornos de producción reales.

La estructura de esta sección es la siguiente. En primer lugar se presentan dos aplicaciones reales que se han construido con el sistema. El objetivo en este punto es proporcionar una visión más clara de la manera en que el sistema puede ser usado para construir aplicaciones en los ámbitos objetivo.

Posteriormente se realiza una evaluación del cumplimiento de los objetivos del sistema, expuestos en la sección I.2, a la luz de los datos recogidos para diversas aplicaciones reales construidas con el sistema.

IV.1. DESCRIPCIÓN DE ALGUNAS APLICACIONES CONSTRUIDAS CON EL SISTEMA

En este apartado se describe cómo se ha utilizado el sistema mediador descrito para solucionar algunos problemas de integración de datos. Se pretende mostrar cómo los conceptos mostrados hasta el momento pueden utilizarse en situaciones reales muy comunes.

De todas las aplicaciones que se han construido con el sistema, se han escogido dos de ellas, una en el ámbito de aplicaciones de búsqueda/comparación/agregación en Internet y otra en el ámbito empresarial. Ambas están funcionando en entornos reales de producción y tratan cargas de procesamiento elevadas.

La primera de ellas es una herramienta para comparar precios entre tiendas electrónicas. Trata un número elevado de fuentes (más de 250) con un grado de autonomía total o casi total.

La segunda aplicación funciona en un entorno corporativo y unifica diversos repositorios de datos heterogéneos dispersos que habían sido generados de manera independiente dentro de una organización empresarial.

Los siguientes subapartados tratan, respectivamente, cada una de estas aplicaciones.

IV.1.1. COMPARADOR DE PRECIOS EN INTERNET

Esta aplicación permite a sus usuarios realizar comparación de precios en Internet de determinados productos.

Desde el punto de vista del usuario, la aplicación funciona de la siguiente manera.

Para cada categoría de producto (se incluyen categorías tales como libros, música, películas en vhs y dvd, hardware, software, juegos, artículos de imagen y sonido, artículos deportivos, teléfonos móviles, canciones en mp3, etc.), existen una serie de tiendas electrónicas asociadas. El total de tiendas supera, en la actualidad, las 250. Cuando un usuario entra al servicio, puede escoger la categoría de producto que desea consultar. En la página asociada a cada categoría, se le presenta un formulario de consulta unificado que le permite consultar a la vez todas el catálogo de todas las tiendas de esa categoría.

Como respuesta de la consulta, el usuario obtiene un listado de los productos acordes con su búsqueda encontrados en todas las tiendas electrónicas. Una vez obtenidos estos resultados, el usuario puede realizar cualquier filtro u ordenación que desee, facilitándole así el proceso de elección de la tienda que le ofrece mejores condiciones. Una vez realizada su elección, puede saltar directamente mediante un enlace a la página del producto en la tienda deseada. En varias de las categorías de

productos (libros, música, películas, etc.) el sistema proporciona también las reseñas de los productos buscados que hayan aparecido en fuentes especializadas, ayudando así a la decisión de compra.

Además, el sistema se encarga por sí solo de determinadas labores de unificación: por ejemplo, las diversas tiendas electrónicas ofrecen sus precios en la moneda local. Sin embargo, el usuario obtiene directamente todos los precios convertidos en su moneda, de acuerdo al cambio del día.

Desde el punto de vista interno, la aplicación está construida de la siguiente manera. Para cada tienda de cada categoría de productos existe una relación base modelando su esquema y las capacidades de consulta permitidas por su formulario de consulta. El programa envoltorio es generado mediante la herramienta de generación semi-automática de programas envoltorio, cuyos fundamentos fueron introducidos en la sección III.4.

Entonces, para cada categoría de productos se define una relación en el esquema global cuyo nombre es el de la categoría de producto y cuyo esquema viene definido por la vista compuesta por la unión de todas sus fuentes asociadas. Por ejemplo, supóngase la categoría de producto `LIBRO`, cuyas tiendas fuesen Amazon, Barnes & Noble, Crisol y Fnac. Supóngase que se ha definido una relación base para cada una de las tiendas, con el mismo nombre que ella, y cuyo esquema consta de los atributos `TITULO, AUTORES, PRECIO` (la aplicación real considera bastantes atributos adicionales). Los dos primeros son de tipo cadena de caracteres, mientras el último es de tipo dinero.

La definición de la relación del esquema global para la tabla `LIBRO` sería:

```
LIBRO= AMAZON ∪ BARNESANDNOBLE ∪ CRISOL ∪ FNAC
```

(NOTA: En realidad, como esta categoría de productos en la aplicación real incluye reseñas, el esquema se construiría haciendo el join entre la unión de las tiendas y la unión de las fuentes de reseñas)

El esquema de `LIBRO` será:

```
LIBRO={TITULO:String, AUTOR:String, PRECIO:Dinero}
```

Sus capacidades de consulta se calculan automáticamente en función de las de las fuentes, tal y como se vio en el apartado III.2.4.1.

Una vez definida la relación `LIBRO`, el sistema permite consultarla directamente en SQL, ocupándose de manera transparente de realizar las subconsultas pertinentes a las fuentes y de integrar los resultados obtenidos. Por supuesto, para que una consulta

pueda ser respondida por el sistema, se precisa que esta sea soportada por las capacidades de consulta de la relación.

Supóngase que las capacidades de consulta obtenidas para LIBRO, una vez minimizadas, pueden representarse mediante los siguientes métodos de búsqueda:

```
{
 NEGATIVAS {
  (TITULO,CONTIENE,1,CUALQUIERA)
 }
 POSITIVAS{
  (TITULO,CUALQUIERA,+,CUALQUIERA)
  (AUTOR,CUALQUIERA,+,CUALQUIERA)
  (PRECIO,CUALQUIERA},+,CUALQUIERA)
 }
 SALIDA {TITULO,AUTOR,PRECIO}
}

{
 NEGATIVAS {
    (AUTOR,CONTIENE,1,CUALQUIERA)
  }
 POSITIVAS {
  (TITULO,CUALQUIERA,+,CUALQUIERA)
  (AUTOR,CUALQUIERA,+,CUALQUIERA)
  (PRECIO,CUALQUIERA},+,CUALQUIERA)
 }
 SALIDA {TITULO,AUTOR,PRECIO}
}
```

En función de las capacidades de consulta permitidas, el creador de la aplicación puede diseñar fácilmente un formulario de consulta web para el usuario sobre LIBRO (de hecho, esta interfaz podría incluso generarse de manera automática). Como puede verse, la única restricción a la que debe amoldarse el usuario es que deberá especificar al menos una condición de consulta, o bien para el atributo TITULO o bien para el atributo AUTOR.

Por ejemplo, un formulario sencillo puede ofrecer una caja de búsqueda para cada atributo, quizás indicando que la caja asociada al atributo PRECIO se usa para especificar un precio máximo. En cuanto a las cajas para los atributos TITULO y AUTOR, podrían permitir escoger mediante una casilla de verificación si se pretende buscar exactamente por el valor especificado en la caja de texto (en cuyo caso debería aplicarse el operador '=')o si se pretende buscar aquellos libros que contengan en el atributo correspondiente la cadena especificada (operador Contiene).

Ahora, dada una consulta del usuario, la aplicación puede muy fácilmente construir una consulta SQL que enviar al mediador. Por ejemplo, si el usuario rellenase dicho formulario asignando el valor 'java' a la caja del atributo TITULO (sin marcar la casilla de 'búsqueda exacta'), el valor 'Peter McNaughton' a la caja del atributo

AUTOR (sí marcando en esta ocasión la casilla de 'búsqueda exacta') y el valor '5000' a la caja del atributo PRECIO, la consulta generada debería ser:

```
SELECT TITULO, AUTOR, PRECIO
FROM LIBRO
WHERE TITULO CONTIENE  'java' AND AUTOR = 'Peter McNaughton' AND
PRECIO<5000
```

Al recibir esta consulta, el mediador se encargará, de la manera que ya se ha visto, de construir los planes de ejecución para la consulta, escoger el más óptimo y ejecutarlo.

El sistema mediador se encarga de aspectos como la unificación de formatos usando las reglas definidas para ello mediante el sistema tratado en el apartado III.2.6 (por ejemplo, para el caso de los nombres de los autores), las conversiones de moneda (manejadas automáticamente por el tipo de dato dinero, en función de información obtenida también de una fuente web, el Banco Central Europeo en este caso) y, a través, de los envoltorios, la consulta directa a través de los formularios de las fuentes y la extracción de los resultados, así como los post-procesados impuestos por el plan de consulta.

También como ya se ha visto, el acceso a las fuentes se realizará en paralelo y la recolección de los resultados se realizará de forma asíncrona. Esto quiere decir que a medida que el mediador vaya obteniendo resultados de las tiendas, estos estarán disponibles inmediatamente para la aplicación, sin necesidad de que esta tenga que esperar a que todas las fuentes terminen. De esta manera, aunque la consulta se realiza en tiempo real a través de Internet, los tiempos de respuesta de cara al usuario son cortos, ya que muy rápidamente empieza a ver resultados en su pantalla.

Una vez obtenidos los resultados, el usuario puede ordenarlos o filtrarlos a conveniencia mediante formularios específicos dispuestos para ello en las páginas de resultados de la aplicación. Estos formularios se traducen en nuevas consultas sobre la tabla LIBRO.

Como quiera que en esta aplicación, el sistema mediador está configurado para guardar en la cache los resultados obtenidos, estas nuevas consultas podrán realizarse ya localmente (de la misma manera si posteriores consultas de otros usuarios pueden responderse en función de los datos obtenidos para esta consulta, antes de que dichos datos expiren, la consulta se resolverá también localmente).

IV.1.2. APLICACIÓN DE UNIFICACIÓN DE REPOSITORIOS CORPORATIVOS

Esta aplicación fue construida para una agencia de noticias que se enfrentaba al problema de integrar todos sus repositorios corporativos.

La organización cuenta con los siguientes repositorios principales en bases de datos:

- Noticias: Contiene noticias.
- Histórico de noticias. Contiene noticias producidas hace más de 3 meses.
- Reportajes: Contiene reportajes de investigación.
- Fototeca: Contiene fotografías en formatos digitales.
- Infografías: Contiene diversos gráficos informativos (por ejemplo, un mapa con información relativa a un tema determinado)
- Multimedia: Contiene ficheros de audio/vídeo.

Todos estos repositorios (excepto los dos primeros) fueron creados en momentos distintos y son gestionados también por equipos de trabajo diferentes. Esto se traduce en que los esquemas, las taxonomías de clasificación empleadas y las tecnologías utilizadas para el almacenamiento, difieren en gran medida.

Por ejemplo, el repositorio de noticias está almacenado en una base de datos relacional de un determinado fabricante. El esquema de esta base de datos hace corresponder a cada noticia un conjunto de temas en los que dicha noticia puede encuadrarse. Esos temas se organizan en forma de un árbol que va siendo extendido por los editores a medida que aparecen nuevos temas. El repositorio de fototeca está almacenado en una base de datos relacional de un fabricante diferente. Su esquema es similar al de la base de datos de noticias, pero su árbol de temas difiere en gran medida. Algo similar ocurre con el resto de repositorios.

Sobre estos repositorios estructurados, los diferentes equipos técnicos han construido aplicaciones de consulta adaptadas a las necesidades particulares de cada caso, que son las que utilizan los redactores y editores de la agencia para consultar y actualizar estas bases de datos.

Sin embargo, no es posible para la organización realizar ninguna consulta que involucre a más de uno de los repositorios. Esto quiere decir, por poner algunos ejemplos sencillos, que no es posible realizar consultas como: 'obtener todos los reportajes del mismo tema que una noticia determinada', 'obtener fotografías que puedan complementar una noticia determinada' o 'obtener todos los items de información disponibles, del tipo que sean, sobre un determinado tema'.

Para afrontar este problema, en un momento dado, la organización planteó una solución de unificación basada en el enfoque de "Base de Datos Universal"; es decir, basada en la construcción de una nueva base de datos unificada. Sin embargo, el coste en tiempo y recursos de dicha solución fue evaluado como excesivo debido a las siguientes causas (que ya se expusieron, de manera general, en el apartado II.3.3.1), por lo que el proyecto fue desechado:

- Dado que las aplicaciones utilizadas en la agencia estaban construidas sobre las viejas bases de datos, era necesario rehacer estas de manera casi total, cuando en realidad el grueso de la funcionalidad requerida seguía siendo el mismo.

- Como consecuencia de lo anterior, el personal tendría que ser formado en el uso de las nuevas herramientas. Esto debe hacerse incluso para aquellos usuarios que no precisan utilizar las nuevas funcionalidades, que eran la gran mayoría.
- Se requiere la adquisición de nuevo y potente hardware y software para soportar el nuevo sistema, que soportaría un importante subconjunto de las operaciones críticas de la empresa y manejará un gran volumen de datos.

El enfoque empleado para abordar el proyecto utilizando el sistema mediador se describe a continuación.

Se comenzó por la construcción de un envoltorio para cada una de las fuentes. Para cuatro de las fuentes se utilizó un envoltorio de tipo JDBC. Para las otras dos fuentes se utilizó un envoltorio de tipo web, que accedía a un formulario de consulta disponible en la Intranet de la agencia. Cada relación base exportada por cada fuente fue construida de acuerdo a la siguiente estructura:

```
R={ID:String, RESUMEN:String, CONTENIDO:String, FECHA:Date,
TEMA:STRING, TIPO_ITEM:Enumerated (NOTICIA, HISTORICO,
REPORTAJE, FOTO, INFOGRAFIA, MULTIMEDIA)}
```

El atributo `ID` es construido para cada relación concatenando el nombre de la misma al comienzo de los identificadores utilizados internamente por dicha relación. El atributo `TIPO_ITEM` toma un valor fijo para todos los items de cada fuente.

Para afrontar las dificultades planteadas por la heterogeneidad de las taxonomías empleadas para clasificar los items de información, se utilizó el mecanismo de tratamiento de heterogeneidades mostrado en la sección III.2.6.1. Con ello se consiguió unificar las taxonomías hasta el cuarto nivel de los respectivos árboles de temas.

Una vez realizadas estas tareas, el sistema está listo para contestar consultas que integren la información de las fuentes. Por ejemplo, para obtener todos los reportajes del mismo tema que una noticia determinada, podría hacerse la consulta:

```
SELECT *
FROM REPORTAJES, NOTICIAS
WHERE NOTICIA.TEMA=REPORTAJE.TEMA AND NOTICIA.ID='idNoticia'
```

dónde `idNoticia` es el identificador de la noticia.

Para tratar todos los contenidos unificadamente, independientemente de su tipo, puede definirse una relación global (a la que llamaremos, por ejemplo, `ITEM_INFORMACION`) que sea la unión de todas las relaciones base. De esa manera podrán efectuarse fácilmente consultas como 'obtener todos los items de un tema determinado', consultando directamente esta vista.

```
SELECT *
FROM ITEM_INFORMACION
WHERE TEMA='tema'
```

donde 'tema' es el tema deseado.

La realización del proyecto con este enfoque no requiere rehacer las viejas aplicaciones ni volver a formar al personal que no necesita utilizar las nuevas funcionalidades. Además, el hardware preciso tiene un coste bajo, dado que el mediador sólo se ocupará de las tareas de integración de información y los datos permanecen en las fuentes originales.

Además de estos repositorios, fundamentalmente estructurados, existen varias fuentes de datos semi-estructurados que el personal de la agencia utiliza habitualmente. Entre ellas se incluyen los sitios web de los medios de comunicación que son sus clientes y reelaboran sus noticias. Esto les permite, entre otras cosas, realizar un seguimiento del uso de sus noticias por parte de los medios, así como comprobar qué porción de las noticias publicadas provienen de su agencia y qué parte proviene de la competencia.

Una manera mucho más eficiente de realizar esta tarea es utilizar un sistema mediador para consultar estas fuentes web como si sus datos estuviesen contenidos en una base de datos local. Esto se ha podido realizar fácilmente mediante la construcción de un envoltorio para cada una de estas fuentes web y la definición posterior de una relación global que une las relaciones base que representan a cada una de las fuentes.

IV.2. EVALUACIÓN DE CUMPLIMIENTO DE OBJETIVOS

En este epígrafe se repasan los objetivos expuestos en el apartado I.2, y se evalúa el grado de cumplimiento de los mismos por parte del sistema mediador.

1) Las aplicaciones construidas con el sistema mediador deben ser capaces de funcionar en entornos de producción reales, soportando las cargas de trabajo que se producen en dichos entornos. En particular, debe considerar las necesidades de aplicaciones Internet accesibles a una audiencia multitudinaria, así como las que son generadas por una aplicación utilizada en una gran corporación para propósitos de negocio. Esto involucra que el sistema debe utilizar una arquitectura distribuida que permita escalar fácilmente la arquitectura hardware y de comunicaciones necesaria.

Algunas de las diversas aplicaciones construidas con el sistema mediador para entornos Internet han estado y siguen estando disponibles en portales de alta audiencia. Alguna de estas aplicaciones sirve en la actualidad más de 1.000.000 de consultas mensuales, utilizando arquitecturas hardware habituales en la industria para ese volumen de consultas. En comparación con otras aplicaciones habituales en sitios web en Internet, para un número similar de consultas, los servicios construidos con el mediador consumen recursos hardware muy similares.

En particular, la siguiente tabla muestra el número de consultas recibidas cada mes, y la media de consultas simultáneas de pico diarias, según datos recogidos durante Septiembre y Octubre de 2001 para una de las instalaciones de la aplicación de compra comparativa descrita en el apartado IV.1.1, accesible desde diversos portales en Internet. El servicio es ofrecido desde un conjunto de tres máquinas Linux con 512 Mbytes de RAM cada una y procesadores funcionando a 300 Mhz.

	Consultas mensuales	**Pico de consultas**
Septiembre 2001	1.112.347	16,1
Octubre 2001	1.193.332	16,12

En cuanto a los requerimientos de comunicaciones, la naturaleza misma de las aplicaciones de comparación/agregación/búsqueda en Internet, hace que el ancho de banda consumido sea considerablemente mayor que en otras aplicaciones, ya que existe un importante volumen de tráfico que circula entre las fuentes y el mediador. Sin embargo, hay que decir que este consumo se ha mostrado siempre dentro de los límites aceptables, debido en gran parte, a la alta eficacia que el sistema de cache presenta en este tipo de entornos.

Más concretamente, en el sistema evaluado, muy representativo de este tipo de aplicaciones, cada consulta involucra, de media, a 8,3 fuentes web (tiendas electrónicas en este caso) y descarga 1,25 páginas por fuente. Esto lleva a que una consulta completa que no utilice la cache, requiera aproximadamente de la descarga de

415 Kbytes de información desde Internet al mediador (considerando una media de 40 Kbytes por página HTML).

Por ello, es muy importante disponer de un sistema que materialice localmente la mayor cantidad de información posible. Sin embargo, nuevamente, esta aplicación concreta vuelve a presentar un escenario desfavorable para la realización de varios esquemas de materialización. Esto se debe a las siguientes razones:

- Las tiendas electrónicas no permiten, normalmente, acceder a todos los elementos de su catálogo, sino que estos sólo pueden ser accedidos a través de formularios de búsqueda en línea.
- Incluso aunque la realización de pre-cargas periódicas fuese posible, el volumen de datos sería demasiado grande, dada la enorme amplitud de los catálogos en línea de las tiendas electrónicas. Ese esquema de funcionamiento sería análogo al de 'Almacén de Datos', con todos los graves inconvenientes asociados a ese enfoque en este tipo de aplicaciones.

Por lo tanto, el esquema de materialización que es necesario seguir en este caso es el de funcionamiento en modo 'cache', donde los datos que van siendo descargados para responder a las consultas de los usuarios, se guardan localmente (hasta que expiran) para contestar a consultas posteriores.

Este esquema se muestra, en la práctica, muy efectivo para las aplicaciones del ámbito Internet. La razón es que, la elevada audiencia de estos servicios proporciona la "masa estadística" suficiente para que haya un gran número de consultas repetidas. La capacidad del sistema para responder consultas particulares en función de consultas más generales acentúa esta tendencia.

Considérese el ejemplo de la comparación de precios de discos compactos musicales. El sistema permite realizar consultas por el título del disco compacto y por su artista. Hay varias razones por las que un sistema de cache como el utilizado en el sistema mediador presentado en este trabajo es altamente efectivo:

- Un porcentaje amplio de los usuarios buscará discos compactos que estén de actualidad o que hayan sido realizados por artistas que estén de actualidad. Por lo tanto, el sistema cache rápidamente almacenará la información sobre estos items, con lo que sólo las consultas de los primeros usuarios requerirán del acceso a las fuentes.
- El número de artistas populares es relativamente reducido. Sin embargo, la inmensa mayoría de las consultas involucran únicamente a estos artistas. Cada vez que un usuario realice una consulta por uno de estos artistas (e.g. "obtener todos los discos compactos del artista 'u2' "), el sistema descargará toda la información disponible en las fuentes sobre discos compactos de ese artista. Cualquier consulta posterior sobre discos compactos de ese mismo artista, podrá ya ser contestada por el sistema cache.

La siguiente tabla muestra la eficacia media del sistema cache en la aplicación de comparación de precios para los principales productos (esto es, aquellos que concentran la mayor parte de las búsquedas), así como la media para el total de productos:

Producto	**% Aciertos en cache**
Libros	44%
Música	58%
Películas	50%
Mp3	60%
Fotos	52%
Hardware	31%
Media de todos los productos	47%

Como puede verse, la cache permite que casi la mitad de las consultas puedan ser resueltas sin acceder a las fuentes. Además, a medida que el número de consultas sobre el sistema aumenta, lo hace también el porcentaje de aciertos. De hecho, aquellos dominios con un porcentaje menor de aciertos son aquellos que reciben menos consultas.

En cuanto a las aplicaciones en el ámbito corporativo, en general ofrecen condiciones más favorables. La principal diferencia es que los anchos de banda disponibles entre el mediador y las fuentes son mucho mayores y más predecibles. Además, el volumen de datos a transferir será normalmente menor debido a que el número de fuentes también lo es y a que dichas fuentes suelen ofrecer sus datos en formatos con una sobrecarga mucho menor (nótese que una página de una fuente web comercial puede ocupar unos 40 Kbytes y contener menos de 2 Kbytes de información útil a extraer). Esto disminuye enormemente las latencias de red y, por lo tanto, la importancia del modo de funcionamiento asíncrono y de la efectividad de la cache es menor.

2) El sistema mediador debe ser capaz de permitir la integración de todos los tipos de fuentes estructuradas y semi-estructuradas habituales tanto en la construcción de aplicaciones Internet como en la construcción de aplicaciones corporativas de integración de datos. Esto incluye, entre otras, fuentes web con su salida en HTML, documentos XML, bases de datos relacionales, hojas de cálculo, documentos de texto semi-estructurado (en formatos tales como PDF, o MS Word), etc.

Esto involucra que los modelos utilizados deben ser lo suficientemente expresivos para reflejar las peculiaridades de cada tipo de fuente en aspectos como: capacidades de consulta, formatos de representación, mecanismos de ejecución, etc

Las aplicaciones construidas hasta el momento con el sistema mediador tratan en la actualidad más de 600 fuentes diferentes, distribuidas de la siguiente manera en función de su tipo:

Tipo de fuente	Número de fuentes
Fuentes web	485
Bases de datos relacionales	24
Hojas de cálculo	14
Ficheros de texto semi-estructurados	42
Fuentes XML	26
Otras	17
Total	608

Cómo puede verse, hay un claro predominio de las fuentes web. En gran medida, la razón es que las aplicaciones en entorno Internet suelen acceder a un número de fuentes muy alto (por ejemplo, la aplicación de compra comparativa ya comentada utiliza unas 250 fuentes). También ha influido la importancia y expectación que este tipo de aplicaciones han alcanzado en los últimos años, lo que ha hecho que la mayor parte de aplicaciones realizadas hasta la fecha con el sistema mediador se hayan realizado en dicho ámbito.

Sin embargo, también puede verse que se han tratado exitosamente fuentes de todos los tipos que figuran entre los objetivos del trabajo. Hasta la fecha, ha sido posible siempre añadir a una aplicación construida con el sistema una fuente de cualquiera de estos tipos.

Un aspecto que es importante mencionar es que gran parte de las fuentes presentan la posibilidad de ser consultadas por varios operadores (no sólo el '='), y de recibir varias condiciones de selección para un mismo atributo dentro de la misma consulta. Ya se comentó en apartados previos que la mayoría de los sistemas mediadores (muchos de ellos basados en el modelo de consultas conjuntivas) no pueden representar adecuadamente estas características.

En concreto, del total de fuentes de las aplicaciones construidas hasta el momento con nuestro sistema, más del 80% utilizan operadores distintos del '=', siendo los más habituales: el operador `CONTIENE` entre cadenas de caracteres y los operadores de comparación. En cuanto a la multiplicidad, más del 62% de las fuentes han presentado una multiplicidad distinta de uno para algún atributo.

3) El sistema mediador debe conseguir proporcionar sobre los datos semi-estructurados una visión que permita manejarlos de manera similar a si estuviesen contenidos en una base de datos convencional (i.e si fuesen datos estructurados).

Tal y como ya se ha comentado sobre el objetivo 2), el modelo de relación utilizado en el sistema ha sido suficientemente adecuado para incluir en el sistema todos los tipos de fuentes semi-estructuradas que se pretendía. Una vez, que estas fuentes están accesibles en el sistema como relaciones base, ya se ha visto que es posible consultarlas con un lenguaje de consultas estructurado, muy similar a SQL, y también

combinarlas entre sí de manera muy similar a como se haría en una base de datos relacional.

4) El sistema mediador debe poder ser integrado de manera sencilla en los entornos de producción habituales actualmente en la mayor parte de centros de procesamiento de datos. Esto involucra que el sistema debe ser acorde con las principales normas existentes actualmente en la industria para el acceso y manejo de datos, tales como SQL o JDBC.

En el momento de escribir estas líneas, el sistema mediador ha sido integrado ya en más de 10 entornos de producción reales diferentes.

El hecho de que el sistema mediador esté implementado íntegramente en lenguaje JAVA (con la única excepción del sistema de navegación en sitios web, que al estar implementado utilizando Microsoft Internet Explorer, debe funcionar en entorno Windows), ha permitido su funcionamiento directo en plataformas como Sun Solaris, IBM AIX, Microsoft Windows y Linux. Además, al sustentar sus mecanismos de distribución sobre la norma Java RMI, su integración con cualquier servidor de aplicaciones J2EE es inmediata. Para el caso de plataformas Microsoft DCOM, cuyo mecanismo de distribución en incompatible con Java RMI, se han utilizado diferentes "puentes" (o *bridges*) comerciales entre DCOM y RMI, con lo cual esto tampoco ha supuesto un problema en la práctica.

Por otro lado, al poder realizarse su acceso a través de la norma JDBC, de amplia difusión en la industria, y ser su lenguaje de consulta muy similar a SQL, los programadores de aplicaciones rápidamente son capaces de integrar el sistema mediador con sus aplicaciones. Por ello, el tiempo de integración del sistema mediador en los entorno de producción, no ha sido significativamente superior al habitual para cualquier otra aplicación de manejo de datos, y las pequeñas diferencias encontradas, pueden atribuirse a la novedad de los aspectos específicos de los sistema mediadores, tales como la configuración de la cache o la configuración de la accesibilidad a las fuentes.

Si bien cada entorno de producción presenta características muy diferentes de las de los demás, que hacen que la realización de medias de los tiempos de instalación y configuración por entorno sean de un valor relativo (la varianza de la distribución es muy elevada y el número de datos recolectados es hasta la fecha, pequeño, en términos estadísticos), la media actual es inferior a los 3 días/hombre.

5) La creación del esquema global unificado de una aplicación construida con el sistema mediador, así como la configuración del mismo, debe poder ser realizada de manera sencilla por cualquier persona con conocimientos básicos de administración de bases de datos convencionales.

Tal y como ya se ha expuesto, las relaciones del esquema global se definen mediante vistas sobre las relaciones base expresadas en un lenguaje muy similar a

SQL. La diferencia más significativa es el tratamiento de las capacidades de consulta de las fuentes. Si bien el sistema calcula por sí sólo las capacidades de las relaciones del esquema global, el administrador debe, previamente, haberlas definido adecuadamente para cada relación base.

Otra diferencia, habitualmente menos importante, son las discrepancias en cuanto a formatos de representación, lo que puede obligar a definir y mantener reglas como las tratadas en el apartado III.2.6.

Creemos que los mecanismos que se han definido en este sistema para el tratamiento de ambos aspectos son lo suficientemente sencillos para ser aprendidos sin dificultad por administradores de bases de datos convencionales o personal con una cualificación similar. Nuestra experiencia hasta la fecha corrobora esta aserción.

6) El mantenimiento de los envoltorios utilizados, incluidos los de las fuentes web, para las fuentes utilizadas por las aplicaciones generadas con el sistema mediador debe poder ser realizado, en un porcentaje superior al 90%, por personal sin capacidades de programación. Esto incluye la generación de envoltorios para nuevas fuentes, así como el mantenimiento de los envoltorios para fuentes ya existentes.

Tal y como ya se ha comentado, este es uno de los aspectos clave que el sistema debe cumplir, especialmente para las aplicaciones que operan fundamentalmente con fuentes web comerciales, que son complejas, cambian con frecuencia y tienen un autonomía elevada, cuando no total, con respecto al sistema mediador.

Como ya se ha dicho también, las aplicaciones construidas con el sistema, manejan en el momento de escribir estas líneas unas 600 fuentes, de las cuales unas 480 son fuentes web. Como ejemplo, el trabajo de mantenimiento y adición de fuentes de la aplicación de compra comparativa (unas 250), así como el proceso de añadir nuevas, es realizado por un equipo de dos personas sin conocimientos de programación. Este equipo está supervisado (con dedicación a tiempo parcial a esta tarea) por una persona que sí tiene habilidades de programación y que se encarga de:

- tareas de coordinación,
- la puesta en marcha de pequeñas mejoras en las herramientas que utiliza el equipo,
- labores que requieren programación o la definición de expresiones regulares complejas. Estas son: la definición de juegos de marcas de formato (ver apartado III.4.3.3.1.2), la definición de nuevas acciones para el sistema de generación de analizadores (ver apartado III.4.3.3.1.5) y la definición de nuevas funciones para el tratamiento de heterogeneidades (ver apartado III.2.6.1.1).

Una vez construidos un conjunto de juegos, acciones y funciones de transformación básicas, las fuentes que han requerido de la programación de nuevas acciones o funciones o la definición de nuevos juegos de marcas, han sido menos del 3% del total. El resto de fuentes han podido ser tratadas por el personal no programador.

El rango de fuentes tratado es amplio y comprende fuentes de una amplia variedad de dominios: tiendas electrónicas, sitios de empleo, portales especializados, periódicos electrónicos, bancas electrónicas, buscadores, bancos de imágenes en línea, sitios de música en línea, sitios de subastas, sitios de información financiera y un amplio etcétera.

En cuanto a las fuentes de otros tipos, la definición de envoltorios para fuentes XML y JDBC es a menudo totalmente automática. En algunos casos, puede requerir escribir *consultas patrón* en SQL (ver apartado III.3.1) o en XQuery (ver apartado III.3.3). Este tipo de fuentes se encuentran en ambientes corporativos, donde el sistema es manejado normalmente por personal de departamentos técnicos.

7) Los tiempos de generación y mantenimiento de programas envoltorio para fuentes web deben ser cortos. Este trabajo se plantea el objetivo de que el esfuerzo medio de creación de un programa envoltorio para una fuente, incluyendo las fuentes web, sea inferior a 1 día de trabajo de una persona sin habilidades de programación. Esto permite tiempos muy cortos para la creación y puesta en marcha de aplicaciones de integración de datos.

La siguiente tabla muestra los esfuerzos medios por acción de mantenimiento sobre las más de 300 fuentes web manejadas por las diversas aplicaciones del sistema, para los meses de Septiembre, Octubre y Noviembre del 2001.

Una acción de mantenimiento puede ser: 1) la adición de una nueva fuente mediante la creación de su envoltorio correspondiente 2) la actualización de un envoltorio, debido a cambios producidos en la fuente. Cada acción es tratada por un miembro del equipo de mantenimiento, sin conocimientos de programación.

Agosto 2001	3,57 horas
Septiembre 2001	3,48 horas
Octubre 2001	3,2 horas

Como puede verse, los tiempos medios de tratamiento son inferiores a 4 horas por acción.

Un punto crucial para la consecución de estos tiempos es la generación de secuencias de navegación mediante el método expuesto en el apartado, basado en el uso de comandos emitidos sobre un explorador comercial de Internet, representando las acciones realizadas por un usuario del navegador para realizar la secuencia. Para fuentes en las que se usan mecanismos complejos de establecimiento de sesión, Javascript, DHTML, etc., la generación de estas secuencias pasa de ser una muy dificultosa labor incluso para programadores expertos, a una labor mecánica incluso para usuarios sin conocimientos de programación.

El empleo de este tipo de técnicas es cada vez más común en fuentes web comerciales. Por ejemplo, prácticamente todas las bancas electrónicas las utilizan. Del

total de fuentes web tratadas hasta el momento por el sistema, las fuentes con este tipo de dificultades suponen en la actualidad algo más del 20% del total.

Antes del desarrollo de esta nueva técnica para la generación de secuencias de navegación, el tiempo medio de una intervención sobre este tipo de fuentes se elevaba a aproximadamente dos días/hombre de un programador experto. Aproximadamente el 75% de su trabajo se invertía en vencer estas dificultades.

Gracias a la inclusión de esta técnica, actualmente los tiempos medios de intervención para estas fuentes son similares a los del resto de fuentes, y pueden también ser realizadas por personal sin conocimientos de programación.

8) El sistema debe ser capaz de funcionar consultando las fuentes en tiempo real, devolviendo así los datos totalmente actualizados. Sin embargo, por propósitos de eficiencia, debe permitir también, siempre a discreción del administrador del sistema, guardar caches o copias locales de parte de los datos de las fuentes, para las aplicaciones para las que esto sea aconsejable.

En su funcionamiento con la cache desactivada, el sistema mediador accede en tiempo real a las fuentes para cada consulta, proporcionando así siempre la información totalmente actualizada. Este funcionamiento se corresponde con un esquema *virtual* puro.

El sistema cache (descrito en profundidad en el apartado III.2.5.6), permite funcionar con un enfoque mixto, en el que para todas o algunas de las relaciones del sistema, se mantiene una cache aprovechando los resultados obtenidos para consultas previas. El hecho de que la cache sea configurable a nivel de relación permite un alto nivel de flexibilidad, permitiendo escoger para cada relación individual la política de materialización más adecuada.

El sistema cache también permite funcionar en modo totalmente *materializado* cuando las capacidades de consulta de las fuentes lo permitan. Para ello, basta efectuar la consulta `select * from R` las relaciones `R` que se desee materializar totalmente. El funcionamiento de la cache hará que todas las tuplas de la relación sean cargadas en la base de datos local.

La implementación actual del sistema incluye una herramienta que permite planificar temporalmente tareas compuestas de consultas. Esta herramienta puede ser utilizada para materializar información seleccionada automáticamente a intervalos periódicos.

En particular, tal y como ya se ha comentado, en las aplicaciones de ámbito Internet muchas veces no es posible materializar previamente relaciones base. A cambio, los porcentajes de acierto de la cache para estas aplicaciones suelen ser muy altos.

En las aplicaciones corporativas, las fuentes presentan normalmente un grado de autonomía mucho menor, lo que facilita la adopción de estrategias de materialización selectivas, materializando los datos de aquellas fuentes que cambian con poca frecuencia y cuyo volumen es lo suficientemente manejable para poder realizarse a intervalos menores que la frecuencia habitual con la que se produzcan cambios importantes.

9) El sistema debe proporcionar algún mecanismo suficientemente flexible para tratar con las heterogeneidades de formatos de representación entre fuentes de datos.

El mecanismo proporcionado por el sistema para tratar las heterogeneidades en los formatos de representación de las fuentes se muestra en la sección III.2.6. Los problemas tratados de manera más habitual se refieren a la unificación de taxonomías dispares entre las fuentes.

Hasta la fecha, estas heterogeneidades han podido ser tratadas de manera suficientemente satisfactoria, si bien en algunos casos ha sido necesario construir listas de reglas bastante largas y tediosas de mantener. De hecho, en ocasiones en las que la aplicación lo permitía, se ha optado por un grado de unificación menor en aras de conseguir una facilidad de mantenimiento mayor del conjunto de reglas de transformación.

Creemos que el ámbito del tratamiento de heterogeneidades de representación es uno de los que precisa mayor trabajo dentro de los sistemas mediadores y de bases de datos federadas. Forma parte también de nuestros intereses para trabajo futuro.

V. DISCUSIÓN, CONCLUSIONES Y LÍNEAS DE TRABAJO FUTURO

V.1. DISCUSIÓN DEL SISTEMA MEDIADOR

En este apartado se realiza una discusión de las características del sistema presentado frente al estado actual del arte descrito en la sección II.5.

En primer lugar, consideramos las diferencias a nivel de arquitectura. A diferencia de otros sistemas con similares intenciones, nuestro sistema presenta una arquitectura distribuida, que permite disponer sus módulos en diferentes máquinas, escalando así con facilidad. Además, al obtenerse las capacidades de consulta de las relaciones globales en función de las de las fuentes, es posible usar unos mediadores como fuentes de otros, lo que permite un grado de distribución y flexibilidad aún mayor. Esto es especialmente importante, porque permite adoptar un enfoque incremental en proyectos de integración de gran tamaño.

La arquitectura del sistema propuesto apuesta también por el uso de normas con amplia difusión en la industria, tales como RMI[RMI01] o JDBC[JDB01]. Esto facilita su integración en entornos ya existentes.

Por otro lado, en cuanto a las técnicas empleadas, tal y como se especificó en el apartado II.5, los aspectos técnicos más relevantes en la construcción de sistemas mediadores son fundamentalmente los siguientes:

1) La reformulación de las consultas realizadas sobre las relaciones del esquema global en términos de las relaciones de las fuentes. La especificación y cálculo de limitaciones en las capacidades de consulta de las fuentes, está también incluida en este punto.
2) Sistema de generación de envoltorios.
3) Ejecución y optimización de las consultas. Este apartado incluye también las estrategias utilizadas para la materialización de datos en el mediador.
4) Resolución de heterogeneidades de representación de formatos.

Los siguientes subapartados se ocupan respectivamente de la comparación del sistema propuesto con respecto al estado del arte en cada uno de estos puntos.

V.1.1. REFORMULACIÓN DE CONSULTAS Y CAPACIDADES DE LAS FUENTES

Para la reformulación de consultas, se ha adoptado, como se ha visto ya, un enfoque de *definición de vistas* (o GAV) frente al enfoque de *definición de fuentes* (o LAV).

Pensamos que para un sistema que aspire a integrarse en la industria con facilidad, este es el enfoque más adecuado debido a que:

- Su similitud con la definición de vistas en bases de datos relacionales hace sencilla la formación de administradores de bases de datos convencionales para actuar como administradores de sistemas mediadores.
- Hace más natural y sencillo que el sistema mediador exporte una interfaz basada en JDBC y SQL, facilitando así también su uso y su integración con otras aplicaciones existentes en los sistemas de información empresariales.
- Los algoritmos para la ejecución de consultas son más eficientes que los del enfoque LAV. Como ya se ha visto, los enfoques LAV requieren de técnicas de alta complejidad algorítmica, incluso cuando funcionan con lenguajes de consulta simples como las consultas conjuntivas. También son más eficientes en el enfoque GAV los algoritmos de obtención de capacidades de consulta y ejecución en presencia de limitaciones de las fuentes.
- La principal desventaja del enfoque GAV (su menor modularidad que obliga a reescribir las definiciones de las relaciones del esquema global cuándo se añaden fuentes), no es un problema grave para el sistema propuesto. La redefinición de las relaciones globales es muy sencilla debido al uso de SQL para definirlas declarativamente, con lo que el esfuerzo de redefinición sólo sería significativo si el ratio de inclusión de nuevas fuentes fuese extremadamente alto (varias al día, por ejemplo). La experiencia acumulada durante la investigación llevada a cabo dice que, al menos hoy en día, esos ratios no se producen en la inmensa mayoría de aplicaciones. Además, si se llegasen a producir dichos ratios, probablemente el cuello de botella estaría en la creación de envoltorios y no en la redefinición de las relaciones globales.

También existen diferencias entre el enfoque presentado y el adoptado en otros sistemas que también utilizan la aproximación GAV. El sistema TSIMMIS[GMPQ95] adopta un modelo de datos semi-estructurado, que permite que sus relaciones incorporen de manera inherente las características de este tipo de datos. Por contra, nuestro enfoque apuesta por un modelo de relación básicamente estructurado (relacional pero con soporte para el tratamiento de atributos de tipo registro y array, así como para limitaciones en las capacidades de consulta).

Utilizar un modelo más estructurado de relación tiene como consecuencia negativa que, siendo las fuentes semi-estructuradas en muchos casos, los envoltorios deben hacerse cargo de las complejidades de hacerlas comportarse con un nivel de estructuración mayor, lo cual complica su construcción. Sin embargo, se considera que este problema es superado mediante la disponibilidad de herramientas para generar los envoltorios para fuentes semi-estructuradas de manera semi-automática.

A cambio, usar un modelo más estructurado permite que la consulta y generación de vistas pueda hacerse con un lenguaje muy similar a SQL, lo que facilita la introducción en el mercado, mientras que con modelos semiestructurados es necesario recurrir a lenguajes nuevos y más complejos. Otra ventaja es que, gracias a la mayor

estructuración, los algoritmos de reformulación, ejecución y optimización pueden ser más eficientes.

En cuanto al tema de las limitaciones en las capacidades de consulta, el sistema presentado en este trabajo presenta importantes ventajas con respecto a otros:

- El marco de descripción de capacidades de consulta permite representar características comunes en muchas fuentes y que no son soportadas por otros esquemas. Con respecto a [YLGU99] (el único que permite calcular las capacidades del esquema global), añade la posibilidad de representar fuentes que utilizan operadores distintos del de igualdad, así como la posibilidad de ejecutar simultáneamente varias condiciones de selección para el mismo atributo. Estas funcionalidades son necesarias para poder modelar adecuadamente un gran número de fuentes (más del 80%, tal y como se comentó en el apartado IV.2). Por ejemplo, tal y como se ha resaltado en varios ejemplos a lo largo del texto, además de los operadores de igualdad, operadores como el de contención entre cadenas de caracteres (al que hemos denominado CONTIENE) son extremadamente frecuentes en fuentes web. Otro ejemplo evidente lo constituyen los operadores de comparación (<, <=, >, >=) muy utilizados en todo tipo de fuentes. También es muy importante distinguir entre aquellas fuentes que permiten una o varias condiciones de selección para el mismo atributo, ya que aprovechar esta característica cuando se presente permitirá contestar un mayor número de consultas y, además, realizar otras de manera más eficiente.
- El sistema obtiene automáticamente las capacidades de consulta de las relaciones globales partiendo de las de las fuentes. Esto se traduce en dos ventajas de crucial importancia:
  - El usuario puede saber a priori las consultas que puede ejecutar el mediador, sin necesidad de procesos de prueba-error
  - Un sistema mediador puede, de esta manera, ser fuente para otro sistema mediador. Esto último es muy útil en sistemas complejos, permitiendo adoptar un enfoque incremental en la construcción de grandes proyectos de integración.
  - De los sistemas presentados hasta el momento en la literatura, sólo [YLGU99] presenta también esta característica. Sin embargo, el sistema propuesto es capaz de soportar los modelos de consulta y descripción de capacidades más ricos que ya hemos comentado, mientras que los algoritmos de [YLGU99] no pueden hacerlo.

V.1.2. GENERACIÓN DE ENVOLTORIOS

Como se ha visto ya, esta es una de las áreas que ha recibido más atención hasta el momento.

El problema más difícil aquí es sin duda la creación de envoltorios para fuentes web y de texto semi-estructurado.

En lo que se refiere al problema de construir secuencias de navegación en fuentes web que implementen las subconsultas solicitadas por el mediador, el sistema propuesto presenta una novedosa técnica que permite disminuir enormemente los tiempos de creación de envoltorios para fuentes complejas, independizando al creador del envoltorio de tareas complejas como realizar el proceso de mantenimiento de sesión o tratar con Javascript u otros lenguajes de scripting. Para fuentes complejas, los ahorros de tiempo son superiores al 75%. Los ahorros de coste son aún superiores ya que, además de la disminución de tiempo, la labor puede ser realizada por personal de menor cualificación (no programadores).

La importancia de esta tarea, que había sido ignorada hasta el momento por la comunidad investigadora, había sido ya previamente remarcada por trabajos como [SH01], dónde se decía: "la extracción de información desde fuentes web comerciales no consiste solamente en técnicas inteligentes de análisis sintáctico, sino también en tratar con las dificultades de navegar por páginas que contienen Javascript o HTML dinámico y requieren cookies o contraseñas". Nuestro trabajo es el primero que da respuesta a estas necesidades.

En cuanto a la extracción de tuplas, el sistema propuesto necesita tratar con todo tipo de fuentes, incluso aquellas más complejas (tales como las bancas electrónicas). Por lo tanto, un enfoque basado en técnicas de aprendizaje inductivo no es adecuado para nuestros propósitos.

Esa es la razón de que se haya optado por un enfoque basado en una herramienta gráfica "supervisora" que genera programas escritos en un lenguaje de especificación de programas de extracción, que nos permite mantener la facilidad de uso incluso con fuentes muy complejas.

Como ya se ha visto en el capítulo anterior, la experiencia acumulada durante el proceso de desarrollo de esta tesis doctoral demuestra que este sistema puede ser utilizado por no programadores para crear y mantener en minutos, programas envoltorio para todo tipo de fuentes web.

Otra ventaja adicional de el enfoque propuesto es que, al basarse en la idea de un patrón cuyas ocurrencias son buscadas en todo el documento, no se ve afectado por muchos cambios en las páginas de la fuente que obligarían a rehacer el envoltorio en otros sistemas. Por ejemplo, si la región donde aparecen las tuplas se desplaza a alguna otra posición de la página (por ejemplo, por la inclusión de anuncios publicitarios o de otras secciones), el sistema no se verá afectado. Esto es muy importante en fuentes web comerciales, donde este tipo de pequeños cambios en el entorno de las tuplas son extremadamente frecuentes.

V.1.3. EJECUCIÓN Y OPTIMIZACIÓN DE CONSULTAS

En lo que refiere a la optimización, el sistema propuesto recopila estadísticas de costes de las fuentes de una manera similar a la mostrada en [ACPS96]. Los costes de los posibles planes de ejecución se obtienen recorriendo desde las hojas a la raíz los posibles árboles para una consulta. El coste de cada nodo se obtiene en función de fórmulas de costes similares a las utilizadas en bases de datos relacionales, con la diferencia de que se contempla el hecho de que las fuentes en muchas ocasiones devuelven sus resultados a ráfagas.

Algunos sistemas recientes[STD00][ILWF00] han construido motores de ejecución dinámicos, capaces de cambiar sobre la marcha el plan escogido para una consulta, en función de factores como retardos inesperados de alguna fuente. Ese aspecto forma parte del trabajo futuro en nuestro sistema.

De todas maneras, la experiencia acumulada en esta investigación sugiere que estas mejoras son más importantes en aplicaciones que funcionen en entorno Internet que en aplicaciones corporativas, ya que estas últimas funcionan sobre redes locales, donde los retardos inesperados son mucho menos frecuentes.

Otro aspecto relacionado con la ejecución de consultas es el sistema cache. Frente a sistemas previos que se basaban o bien en el enfoque *virtual* o bien en el *materializado*, el sistema propuesto utiliza un enfoque que permite escoger para cada aplicación la estrategia más adecuada.

V.1.4. TRATAMIENTO DE HETEROGENEIDADES

Hasta la fecha se han realizado pocos trabajos en este difícil ámbito para sistemas mediadores.

El enfoque mostrado en [Coh98], que se basa en la aplicación de técnicas de recuperación de la información para identificar diferentes textos que se refieran a la misma entidad, tiene como ventaja que funciona de manera automática, sin intervención manual.

Sin embargo, a pesar de su interés como primer peldaño en la automatización de estas tareas, no es suficiente en la práctica debido a que sólo es capaz de tratar casos muy simples, en que se pretende identificar una misma entidad expresada de maneras diferentes en las fuentes. Sin embargo, el problema de las heterogeneidades de representación abarca muchos más casos que no son tratables con este enfoque: por ejemplo, no sirve para tratar heterogeneidades en taxonomías.

Incluso dentro del género de problemas que trata, en muchos casos la detección fracasará (por ejemplo, el uso de sinónimos puede despistar fácilmente a este sistema).

Nuestro enfoque apuesta por definir un sistema de reglas que permitan definir de forma declarativa las heterogeneidades de formato esperadas entre las fuentes. Además el sistema es extensible para que pueda ajustarse a necesidades *ad-hoc*. Este enfoque permite solucionar de manera sencilla la gran mayoría de los casos que aparecen en la práctica.

Sin embargo, presenta limitaciones. En ocasiones, la lista de reglas a definir es larga y tediosa de definir y mantener. Creemos que se requiere más trabajo de investigación para la construcción de medios más automáticos de alcance general. Esta tarea forma parte de nuestro trabajo futuro.

V.2. CONCLUSIONES

Este apartado resume las principales contribuciones del trabajo presentado y expone las principales conclusiones del mismo.

V.2.1. RESUMEN DE LAS PRINCIPALES CONTRIBUCIONES

En este apartado se repasan las principales contribuciones de esta tesis doctoral, que fueron introducidas en al apartado I.3, comentándolas en mayor profundidad a la luz de la discusión anterior.

Frente a sistemas previos presentados en la literatura, el sistema mediador presentado en este trabajo presenta las siguientes aportaciones:

1) Un modelo de representación de relaciones y sus capacidades de consulta con la suficiente potencia para tratar las fuentes encontradas en aplicaciones reales, permitiendo tener en cuenta importantes características tales como la posibilidad de utilizar operadores arbitrarios o la representación de las características de multiplicidad de las fuentes (esto es, cuantas condiciones de selección sobre un mismo atributo son capaces de ejecutar en una sola consulta). Este modelo es descrito en el apartado III.2.1.

La mayoría de los sistemas de representación de las capacidades de consulta de una fuente de datos que se han propuesto hasta el momento[YLGU99][Lev00], no permiten representar adecuadamente fuentes que utilicen operadores de consulta diferentes a los de igualdad ni que admitan más de una condición de selección por atributo en una determinada consulta. Esto, por un lado, impide el tratamiento correcto de un gran número de fuentes (más del 80% de las tratadas hasta el momento por nuestro sistema) y, por otro, hace menos eficiente el tratamiento de otras (más del 62%): en particular, se desaprovecha la capacidad de muchas fuentes para ejecutar más de una condición de selección sobre el mismo atributo dentro de la misma consulta, teniendo así que ser el mediador el que implemente las condiciones de selección adicionales mediante el post-procesado de los resultados obtenidos. En [PGMW95] se presenta un sistema que sí es capaz de representar correctamente este tipo de capacidades. Sin embargo, en dicho sistema no se calculan las capacidades de consulta de las relaciones del esquema global, ni es posible representar explícitamente condiciones de consulta obligatorias.

2) Un algoritmo para calcular las capacidades de las relaciones del esquema global en función de las de las fuentes, lo que permite conocer a priori las consultas que pueden ser contestadas por el sistema mediador, además de permitir que un mediador pueda actuar como fuente para otros mediadores. El algoritmo genera además información base que es utilizada por el algoritmo de ejecución de consultas sobre el sistema. Se describe en el apartado III.2.4.

Hasta el momento, sólo en [YLGU99] se había tratado el problema de obtención de las capacidades de consulta de las relaciones del esquema global en función de las de las fuentes. Sin embargo, este sistema utiliza un modelo de representación de capacidades más simple que el utilizado aquí, por lo que no es aplicable, y se ha construido un nuevo algoritmo capaz de tratar con el nuevo modelo.

La escasa atención prestada hasta el momento a este problema nos parece singular, dado que, tal y como se resaltará nuevamente en la sección de conclusiones, nuestra experiencia dice que esta característica es imprescindible para cualquier sistema mediador que pretenda ser utilizado de manera significativa en aplicaciones reales.

3) Un sistema de generación de programas envoltorio que permite a usuarios no programadores, generar en muy poco tiempo envoltorios para fuentes web, incluyendo las complejas fuentes comerciales de hoy en día. El sistema es descrito en el apartado III.4.

Tal y como ya se ha visto en el capítulo IV, el sistema ha sido utilizado para construir aplicaciones reales que, en el momento de escribir estas líneas, utilizan más de 480 fuentes diferentes. Los tiempos medios para añadir y mantener envoltorios para estas fuentes están por debajo de las 4 horas por fuente, realizando este trabajo personal sin conocimientos de programación.

Este sistema cuenta con varias características que lo diferencian de los presentados previamente en la literatura:

- Permite generar las secuencias de navegación a través de sitios web de manera que el creador del envoltorio puede independizarse totalmente de las complejidades asociadas a aspectos como el tratamiento de identificadores de sesión, Javascript, formularios generados dinámicamente, etc.
- Utiliza un lenguaje de definición de especificaciones de extracción de información y na herramienta supervisora que combinan la potencia suficiente para tratar fuentes complejas, con la simplicidad requerida para que el sistema sea utilizado de manera exitosa por personal sin conocimientos de programación.

Además, como ya se ha visto, el sistema incorpora una herramienta gráfica que permite la generación visual de secuencias de navegación y especificaciones de extracción, haciendo aún más rápida y simple la generación de envoltorios.

4) Permite que el administrador del sistema escoja para cada aplicación el esquema de almacenamiento que mejor se ajuste a sus características. Los principales esquemas son: virtual (en el cuál los datos permanecen en las fuentes), materializado (en el cual copias de los datos son guardados localmente por el mediador) o mixto (en el que parte de los datos se guardan localmente y parte permanecen en las fuentes. Esto es logrado mediante un sistema cache que utiliza también un novedoso mecanismo para representar sus contenidos. El sistema cache es presentado en el apartado III.2.5.6.

Como se ha mencionado, el sistema mediador proporciona un alto grado de flexibilidad para escoger el enfoque más adecuado para cada aplicación:

- Puede adoptarse un enfoque *virtual* puro simplemente desactivando el sistema cache. Esto es aconsejable si los datos de todas las fuentes cambian muy frecuentemente y/o se desean respuestas completamente actualizadas.
- Puede adoptarse un enfoque *materializado* puro utilizando el planificador de actualizaciones (ver apartado III.1.2.4.1) para precargar periódicamente los datos de las fuentes. Este enfoque es adecuado si los datos de las fuentes varían con poca frecuencia y/o su volumen permite actualizarlos con una periodicidad adecuada.
- Puede adoptarse un enfoque *mixto* en el que sólo los datos de ciertas relaciones son materializados periódicamente y el resto son accedidos según el enfoque virtual. Esta aproximación es recomendable cuando en una misma aplicación aparecen fuentes en las que son aplicables las consideraciones expuestas en los dos puntos anteriores.
- Puede adoptarse un enfoque *mixto* basado en la idea de *cache*, en la que se van materializando de forma automática los datos de las consultas que se van realizando. Esta estrategia suele ser muy adecuada, por ejemplo, para aplicaciones de búsqueda/comparación en Internet, tal y como ya se explicó en la sección IV.2 Además, puede escogerse el aplicar esta idea para todas las relaciones o sólo para aquellas que se seleccione.
- En cualquiera de los enfoques anteriores, los tiempos de expiración de los items materializados pueden ser decididos para cada relación (y, en particular, para cada fuente que, como se recordará, son representadas por las llamadas *relaciones base*), permitiendo así que se ajusten a las peculiaridades de cada una (cada fuente suele tener un intervalo propio de actualización).

5) Una arquitectura distribuida que le permite escalar fácilmente para el tratamiento de aplicaciones con cargas de trabajo muy altas. Además, su confianza en normas de amplia difusión en la industria, tales como JDBC, SQL o un modelo de datos basado en el relacional, permite su rápida integración en los actuales entornos de producción. También facilita la formación en el uso del sistema. La arquitectura del sistema se describe en el apartado III.1.

La arquitectura del sistema es distribuida y escalable desde varios puntos de vista:

- Desde un punto de vista interno, la arquitectura se divide en módulos, que se conectan entre sí a través de interfaces normalizados y con un amplio apoyo en la industria: por ejemplo, los envoltorios se comunican con la capa lógica a través de Java RMI o la capa lógica se comunica con la base de datos utilizada para el sistema cache a través de JDBC. Esto permite también soportar replicación y distribución de carga de los componentes.

- El sistema es accesible por aplicaciones externas también a través de interfaces normalizadas, como JDBC y RMI. Esto permite su fácil integración en el seno de otras aplicaciones que sean a su vez distribuidas.
- Un sistema mediador puede ser utilizado como fuente para otros sistemas mediadores. Tal y como ya ha sido comentado, esto es posible gracias a que el sistema calcula automáticamente las capacidades de consulta de las relaciones del esquema global en función de las de las fuentes. Esta funcionalidad permite la construcción de jerarquías de sistemas mediadores tan complejas como se desee y capaces de abordar escenarios de integración muy complejos.

V.2.2. CONCLUSIONES OBTENIDAS

En este apartado se exponen las principales conclusiones de este trabajo.

CONCLUSIÓN 1: *Los sistemas mediadores constituyen una muy buena alternativa para la realización de aplicaciones de integración de información en los modernos ámbitos de negocio, caracterizados por un elevado número de fuentes de información, que se encuentran dispersas, son altamente heterogéneas, presentan un fuerte grado de autonomía y pueden presentar características semi-estructuradas. En particular, el sistema presentado en este trabajo cumple los requisitos para servir como base para la construcción de aplicaciones reales en varios importantes ámbitos de aplicación.*

El mundo actual plantea nuevos retos en lo que se refiere a la disciplina de Integración de Datos. Nunca han existido tantas fuentes de información tan fácilmente accesibles. Sin embargo, estas fuentes son muy numerosas, se encuentran dispersas, presentan una estructura débil, un alto grado de heterogeneidad y un nivel alto de autonomía. Un ejemplo perfecto de esta característica del mundo actual es la llamada *telaraña mundial* (o *World Wide Web*).

Una aplicación de integración de información basada en el enfoque de base de datos universal es imposible de aplicar en la inmensa mayoría de estas situaciones:

- En las aplicaciones de búsqueda/comparación/agregación en Internet, este enfoque es obviamente imposible. Las fuentes son autónomas y no es factible clausurarlas y crear una nueva base de datos con los datos de todas ellas.
- En aplicaciones corporativas, este enfoque sigue siendo poco aconsejable en la mayoría de casos: involucra rehacer las aplicaciones que funcionasen sobre las fuentes, volver a formar a sus usuarios, una fuerte inversión en el nuevo sistema centralizado y, además, añadir nuevas fuentes será muy costoso.

Los enfoques usados tradicionalmente para integración de datos, tales como el de almacén de datos (o *Data Warehousing*), no son adecuados para muchos de los nuevos problemas planteados, tal y como ya se discutió en apartados previos. Estos enfoques deben materializar obligatoriamente todos los datos de las fuentes en el

sistema de integración. Esto puede ser imposible o altamente costoso en estas nuevas situaciones:

- En escenarios donde las fuentes son altamente autónomas y con capacidades de consulta limitadas, puede no ser posible materializar previamente todos los datos de una fuente. Algunas fuentes pueden no permitir obtener todos sus datos de manera que estos puedan copiarse en el sistema de integración. Por ejemplo, las tiendas electrónicas en Internet rara vez permiten realizar consultas que devuelvan todo su catálogo, sino que obligan a consultar este por una serie de atributos que restringen los resultados de la búsqueda.
- Incluso cuando la materialización total de los datos sea permitida por las capacidades de consulta de las fuentes, puede seguir siendo imposible o muy inconveniente en la práctica debido a que los volúmenes de datos a transferir pueden ser enormes.
- No permite el acceso a los datos en tiempo real (esto es, actualizados).

Por su parte, los sistemas de bases de datos federadas que comparten con los mediadores las principales ideas arquitecturales y, por lo tanto, no presentan los inconvenientes anteriores, tienen sin embargo otro importante inconveniente: no pueden tratar con fuentes semi-estructuradas y estas son cada vez más abundantes.

Por último, los sistemas de catalogación e indexación automática, aunque muy útiles en el tratamiento de información no estructurada, no son capaces de aprovechar la estructura que sí existe en la fuentes estructuradas y semi-estructuradas, y sólo pueden ofrecer interfaces para la realización de consultas imprecisas (i.e por palabra clave).

Por supuesto, esto no quiere decir que todos estos enfoques de integración de datos no sean adecuados para ninguna aplicación: todos ellos son útiles en diversas situaciones.

Sin embargo, ninguno presenta la flexibilidad de los sistemas mediadores en las situaciones aquí planteadas: un alto número de fuentes, que además están dispersas, son heterogéneas, pueden ser semi-estructuradas, y que presentan un alto grado de autonomía.

Sin embargo, a priori, para la creación de sistemas mediadores existen importantes dificultades. Era necesario demostrar que esas dificultades podían ser vencidas y que esta idea, teóricamente muy atractiva, podía funcionar bien en la práctica.

Nuestra experiencia en el uso del sistema mediador presentado en este trabajo nos lleva a concluir que, efectivamente, esas dificultades pueden ser vencidas de manera satisfactoria.

De hecho, el sistema ha demostrado su adecuación para la construcción de aplicaciones de integración de información en ámbitos como la búsqueda/comparación/agregación en Internet y el de integración de información corporativa para aplicaciones tan estratégicas en la actualidad como la Gestión de

Relaciones con Clientes (CRM) o la construcción de Portales Corporativos (EIP). Estas aplicaciones han podido soportar las altas exigencias de rendimiento, escalabilidad y facilidad de integración planteadas por los modernos entornos de negocio basados en tecnologías telemáticas. Además, incluso en los casos en los que otros enfoques de integración eran también posibles, se han conseguido unos tiempos de desarrollo y unos costes mucho menores.

CONCLUSIÓN 2: *La arquitectura distribuida del sistema mediador presentado en este trabajo permite una adecuada escalabilidad, lo que le hace capaz de soportar aplicaciones con carga elevada, manteniendo buenas prestaciones.*

Tal y como ya se ha comentado, la arquitectura del sistema es distribuida y escalable desde varios puntos de vista:

- Desde un punto de vista interno, la arquitectura se divide en módulos, que se conectan entre sí a través de interfaces normalizados y con un amplio apoyo en la industria. Los diferentes módulos pueden distribuirse en máquinas diferentes y es posible destinar varias máquinas para distribuir la carga de operación de algún módulo (e.g. el módulo de los programas envoltorio).
- Al exponer interfaces de acceso y consulta normalizados como RMI o JDBC, nuestro sistema puede también ser fácilmente integrado en el seno de una aplicación que siga una arquitectura distribuida. Por ejemplo, nuestro sistema ha sido frecuentemente utilizado dentro de la popular arquitectura J2EE.
- Un sistema mediador puede ser utilizado como fuente para otros sistemas mediadores. Tal y como ya ha sido comentado, esto es posible gracias a que el sistema calcula automáticamente las capacidades de consulta de las relaciones del esquema global en función de las de las fuentes. Esta funcionalidad permite la construcción de jerarquías de sistemas mediadores tan complejas como se desee y capaces de abordar escenarios de integración muy complejos.

Nuestra experiencia muestra que estas características son necesarias. Por ejemplo, las aplicaciones de búsqueda/agregación/comparación en Internet están orientadas a servir a un gran número de usuarios y acceder a un gran número de fuentes. Además, cualquier aplicación Internet puede experimentar fuertes y repentinos incrementos de tráfico (por efecto de una campaña publicitaria, por ejemplo), y los sistemas deben poder adaptarse rápidamente y con el menor coste posible a la nueva situación. Estos requerimientos son inabordables sin una arquitectura distribuida.

Las modernas aplicaciones corporativas no son tampoco ajenas a la necesidad de soportar altas cargas de trabajo y hacer frente a cambios repentinos en sus condiciones de operación. De hecho, prácticamente todas las organizaciones grandes están adoptando o han adoptado ya arquitecturas de desarrollo basadas en un paradigma distribuido. Cualquier sistema mediador que pretenda ser utilizado ampliamente en la industria debe apostar también por esta filosofía.

CONCLUSIÓN 3: *La apuesta por el seguimiento de normas con un gran apoyo en la industria se traduce en que el sistema mediador puede ser fácilmente integrado en los entornos de producción reales. Por otra parte, el uso de un enfoque Global As View y de un lenguaje similar a SQL para la creación de la relaciones del esquema global, y el uso de este mismo lenguaje como lenguaje de consultas, se traduce en unos tiempos de formación menores para que los administradores de sistemas de bases de datos puedan administrar sistemas mediadores.*

Tal y como ya fue comentado en la sección IV.2, el sistema mediador ha sido instalado ya en más de 10 entornos de operación diferentes, pertenecientes fundamentalmente a portales de alta audiencia en Internet y a grandes organizaciones.

Nuestra apuesta por la filosofía de desarrollo de aplicaciones basadas en el lenguaje JAVA, ha llevado a que el sistema pueda funcionar en prácticamente todos los sistemas operativos con una presencia significativa en la industria (con la excepción del sistema de navegación por fuentes web, que al basarse en el Microsoft Internet Explorer, sólo puede funcionar sobre plataformas Windows). De hecho, actualmente existen instalaciones del sistema en los siguientes sistemas operativos: Linux, Microsoft Windows NT y 2000 , SUN Solaris e IBM AIX.

Por otra parte, el uso de JDBC como interfaz para el establecimiento de conexiones con el sistema y la ejecución de consultas, así como el uso de RMI para implementar las comunicaciones entre el cliente y el servidor del sistema, han facilitado en gran medida el que los administradores y desarrolladores utilizasen el sistema, ya que pueden utilizar para ello, tecnologías que ya conocen bien.

El mismo razonamiento es aplicable al uso del lenguaje SQL para la definición de las relaciones del esquema global y la ejecución de consultas. Su similitud, respectivamente, con la manera de crear vistas y ejecutar consultas en una base de datos relacional convencional, aceleran la formación de los administradores y usuarios del sistema. La adopción de un enfoque Local As View y de lenguajes de consultas basados en predicados lógicos, sin duda hubiese sido mucho más difícil de asimilar por el personal de los departamentos técnicos de las organizaciones, ya que, al menos hasta el momento, este tipo de lenguajes no han logrado apoyo por parte de la industria y son desconocidos para muchos profesionales.

CONCLUSIÓN 4: *Los modelos de relación utilizados en sistemas mediadores deben permitir la representación de las capacidades de consulta de las fuentes, utilizando un mecanismo lo suficientemente expresivo. En particular, es necesaria la capacidad para expresar qué operadores pueden utilizarse en las consultas de las fuentes, (ya que más del 80% de las fuentes tratadas hasta el momento lo han requerido). También es muy conveniente poder representar si una fuente permite recibir más de una condición de selección para un mismo atributo en una misma consulta.*

La mayoría de los sistemas de representación de las capacidades de consulta de una fuente de datos que se han propuesto hasta el momento, no permiten representar fuentes que utilicen operadores de consulta diferentes a los de igualdad ni que admitan más de una condición de selección por atributo en una determinada consulta.

Estas funcionalidades son necesarias para poder modelar adecuadamente un gran número de fuentes (más del 80% de las tratadas en nuestro sistema). Por ejemplo, tal y como se ha resaltado en varios ejemplos a lo largo del texto, además de los operadores de igualdad, operadores como el de contención entre cadenas de caracteres (al que hemos denominado `CONTIENE`) son extremadamente frecuentes en fuentes web. Otro ejemplo evidente lo constituyen los operadores de comparación (<, <=, >=, >) muy utilizados en todo tipo de fuentes. Los modos de tratar datos semiestructurados y los nuevos tipos de datos que aparecen en fuentes web (e.g. nuestro sistema mediador contempla, por ejemplo, enlaces o referencias) también requieren nuevos operadores.

También es muy importante representar si las fuentes permiten varias condiciones de selección para el mismo atributo, ya que aprovechar esta característica cuando se presente permitirá contestar un mayor número de consultas y, además, realizar otras de manera más eficiente. En las fuentes tratadas hasta el momento, esta funcionalidad ha podido aprovecharse en algo más del 62% de los casos.

CONCLUSIÓN 5: *El cálculo automático de las capacidades de consulta del esquema global en función de las de las fuentes es una característica crucial en un sistema mediador.*

Tal y como ya se ha comentado en otros apartados previos, nuestra experiencia dice que esta característica es imprescindible para cualquier sistema mediador que pretenda ser utilizado de manera significativa en aplicaciones reales ya que:

- Los usuarios quieren saber *a priori* qué consultas puede contestar el mediador. Lo contrario exige realizar procesos de prueba-error para saber qué consultas pueden ser resueltas. Esto es difícilmente aceptable para la mayoría de usuarios.
- Las capacidades de las relaciones globales deben ser conocidas si pretendemos que un sistema mediador pueda actuar como fuente para otros mediadores. Esta característica permite la adopción de un proceso incremental para la integración de información en grandes proyectos.

CONCLUSIÓN 6: *El sistema de generación de envoltorios presentado en este trabajo permite que personal sin conocimientos de programación genere envoltorios para fuentes semi-estructuradas (incluso para fuentes web comerciales muy complejas) a un ritmo superior al de dos fuentes por día de trabajo. Esto permite construir de manera rápida y con costes de mantenimiento bajos, aplicaciones que acceden a cientos de fuentes, incluso cuando estas presentan un grado alto o total de autonomía.*

Como se comentó en el apartado IV.2, el sistema de generación de envoltorios presentado en este trabajo se ha utilizado para construir varias aplicaciones que acceden a gran número de fuentes (más de 600). Como ya se comentó también, el tiempo medio de una intervención por parte de una persona sin conocimientos de programación para añadir o reparar una fuente web es inferior a las 4 horas. Con fuentes de otros tipos, el proceso es habitualmente aún más rápido, cuando no prácticamente automático (como en el caso de fuentes JDBC o de muchas fuentes XML).

CONCLUSIÓN 7*: La construcción de secuencias de navegación utilizando un explorador de Internet, en lugar de con el enfoque de rutas HTTP, es imprescindible si se desea que la creación y mantenimiento de envoltorios para webs comerciales complejas pueda ser realizado en poco tiempo y por personal sin conocimientos de programación. Por otra parte, el lenguaje presentado para la extracción de datos desde fuentes semi-estructuradas es suficientemente general para poder tratar todo tipo de fuentes web y de texto semi-estructurado. Además, el uso de una herramienta gráfica supervisora permite que su uso sea lo suficientemente simple para poder ser aprendido por personal sin conocimientos de programación.*

Ya se han comentado (ver apartado II.5.4.1.1.1) algunas de las grandes dificultades que plantean las modernas fuentes web comerciales: mantenimiento se sesión mediante identificadores numéricos (que pueden estar divididos en porciones que se combinan mediante funciones Javascript), HTTPS, formularios invocados mediante Javascript, DHTML, etc.

Usar algún esquema basado en peticiones HTTP para implementar secuencias de navegación en estas fuentes es una labor de varios días incluso para programadores expertos.

Esto hace que los tiempos precisos para añadir nuevas fuentes y mantener las existentes sean muy altos y, a menudo, inasumibles. Además, al tener que ser realizada la tarea por medio de programadores expertos, el coste es también muy superior.

Un esquema basado en la construcción de secuencias de navegación utilizando un lenguaje que describe las acciones de un usuario sobre su navegador e implementado sobre un explorador comercial, hace totalmente transparentes estos problemas para el creador del envoltorio. Este sistema (presentado en el apartado III.4.4) es el primero con estas características presentado en la literatura sobre mediadores y creación de envoltorios para fuentes web, permitiendo ahorros de tiempo de en torno al 75% para fuentes complejas.

Por otra parte, el lenguaje diseñado para escribir especificaciones de extracción, ha demostrado poder tratar todas las situaciones encontradas en más de 480 fuentes web y de texto semi-estructurado. Además, los envoltorios son construidos en más del 97% de los casos por personal sin conocimientos de programación.

CONCLUSIÓN 8: *Es necesario que los sistemas mediadores proporcionen flexibilidad para escoger un enfoque virtual, materializado o mixto, dependiendo de las necesidades concretas de cada aplicación.*

Diferentes aplicaciones requieren diferentes estrategias de materialización. Los datos de algunas fuentes cambian tan frecuentemente y la importancia de que la información esté totalmente actualizada puede ser tan grande, que es imprescindible utilizar con ellas un enfoque puramente virtual.

En otros casos, los datos de las fuentes cambian con poca frecuencia y/o el volumen de datos que contienen es lo suficientemente manejable para realizar pre-cargas completas a intervalos periódicos.

Para otras aplicaciones en las que no es posible o aconsejable realizar pre-cargas completas, un enfoque de materialización basado en la idea de cache puede mostrarse muy efectivo, tal y como se demostró en el apartado IV.2.

Es necesario que los sistemas mediadores que aspiren a ser utilizados para la construcción de un rango amplio de aplicaciones, permitan el funcionamiento con cualquiera de estas estrategias y permitan, además, fijar esta estrategia de manera independiente para cada fuente en cada aplicación.

CONCLUSIÓN 9: *El sistema de cache presentado permite conseguir porcentajes de acierto muy elevados para las aplicaciones de búsqueda/comparación en Internet y también elevados para el resto de aplicaciones.*

Las aplicaciones de búsqueda/comparación en Internet se benefician del hecho de ser utilizadas por una alta audiencia. El número de consultas distintas realizadas por los usuarios es comparativamente pequeño, por lo que los porcentajes de acierto conseguidos son normalmente muy altos. En el apartado IV.2 ya se discutió este aspecto con mayor profundidad y se ofrecieron datos de aplicaciones reales que obtenían porcentajes de acierto cercanos al 50%.

Otras muchas aplicaciones en otros ámbitos pueden también beneficiarse de un enfoque de materialización basado en la idea de cache. Las situaciones en las que consultas iguales o similares se repiten con frecuencia son muy habituales. El sistema presentado en este trabajo permite disponer de una estrategia de este tipo, eficiente (los datos en la cache pueden estar almacenados en una base de datos relacional) y que es capaz de identificar que una consulta puede ser contestada con datos de la cache, aunque esa consulta nunca se haya producido antes en su forma exacta.

V.3. LÍNEAS DE TRABAJO FUTURO

Nuestras líneas de trabajo futuro se centran en cuatro aspectos fundamentales:

- Construcción de modelos más ricos para la representación de capacidades positivas y de algoritmos capaces de tratarlos adecuadamente
- Motores adaptativos para la ejecución y optimización de consultas,
- Introducción de técnicas de aprendizaje inductivo para automatizar el proceso de creación de envoltorios tratando de abarcar incluso fuentes complejas, y
- Investigación de nuevos métodos más automáticos para tratar la heterogeneidad de formatos de representación entre las fuentes.

En cuanto al primer aspecto, si bien el modelo de representación de capacidades de consulta presentado en este trabajo se ha mostrado suficientemente expresivo para tratar las fuentes encontradas, existe todavía espacio para su mejora.

En concreto, este margen de mejora se encuentra en la descripción de las capacidades positivas de las fuentes, es decir aquellas capacidades que permiten que el mediador delegue a las fuentes ciertas operaciones del plan de consultas, optimizando así el procesamiento.

Al igual que el resto de sistemas presentados hasta el momento, el marco de descripción de capacidades presentado en este trabajo permite representar la capacidad de las fuentes para realizar operaciones de selección.

La operación de selección es quizás la que mayor beneficio ofrece cuando es delegada a las fuentes, ya que permite disminuir mucho la cantidad de datos a transferir por la red entre la fuente y el mediador.

Sin embargo, algunas fuentes, cuyo principal exponente son las bases de datos relacionales, son capaces de ejecutar por sí mismas operaciones n-arias de álgebra relacional, como el join o la unión. Por ejemplo, si una fuente JDBC exporta dos relaciones que se corresponden con dos tablas en dicha fuente, y una consulta determinada requiere hacer el join entre dichas tablas, entonces sería posible delegar dicha operación a la fuente, optimizando así la carga de procesamiento en el mediador. Sin embargo, para que ello sea posible, sería necesario que las fuentes pudiesen representar esa capacidad mediante algún modelo, asíc omo desarrollar algoritmos de ejecución y optimización de consultas, capaces de aprovechar convenientemente esta información.

En cuanto al segundo aspecto, ya se ha comentado que sería conveniente la realización de optimizadores y motores de ejecución que fuesen capaces de reaccionar dinámicamente a cambios inesperados en los costes de las fuentes. Por ejemplo, si una fuente está congestionada en un momento dado, los costes reales de consulta sobre esa fuente serán muy superiores a los teóricos y sería conveniente que el sistema pudiese reaccionar dinámicamente ante ello. Lo mismo puede ocurrir a causa de un problema de congestión de red.

Nuestra experiencia sugiere que, en la actualidad, esta funcionalidad no es primordial en entornos corporativos, porque en la mayoría de estos casos, las fuentes y el sistema mediador se encuentran dentro de la misma red local o en redes locales próximas, con lo que los retardos de red son bajos, y el comportamiento de la red y las fuentes es más predecible.

En el caso de aplicaciones en el entorno Internet, sin embargo, los retardos de red son mayores y mucho más impredecibles. Además el grado de autonomía de las fuentes es mucho mayor, con lo cual su comportamiento desde el punto de vista del sistema puede ser también mucho más inestable. En estas aplicaciones, por tanto, los beneficios de un esquema de ejecución adaptativo se hacen más evidentes.

En cuanto a la generación de envoltorios para fuentes web, ya se han discutido los problemas de los algoritmos de aprendizaje inductivo para funcionar adecuadamente con fuentes complejas. Sin embargo, también se han hecho constar sus ventajas potenciales ya que, con aquellas fuentes con las que funcionan correctamente, requieren una intervención humana menor y representan la promesa de un mantenimiento automático de los programas envoltorio cuando las fuentes cambien. Pensamos que nuestra experiencia tratando con fuentes muy complejas puede permitirnos colaborar en la extensión de estos métodos para que sean capaces de tratar progresivamente con fuentes más complejas.

En lo que se refiere al cuarto y último aspecto, ya se ha comentado que el método utilizado por el sistema mediador para tratar la heterogeneidad de formatos de representación entre las fuentes es suficientemente potente en cuanto a su capacidad expresiva, pero puede llevar a veces a la definición de conjuntos de reglas largos y tediosos de construir y de mantener. Planeamos estudiar métodos que permitan la generación automática o semiautomática de estas reglas. Ya se ha realizado algún trabajo en este sentido para el caso concreto de unificación de las taxonomías utilizadas en formularios de búsqueda web.

Por otro lado, pretendemos que el enfoque de nuestro trabajo siga siendo eminentemente práctico. El foco de nuestro trabajo seguirá siendo ajustarse lo más posible a las necesidades de las aplicaciones demandadas por la industria. Pretendemos que nuestra experiencia implantando soluciones reales nos permita guiar la evolución de nuestro sistema, y de nuestro trabajo en general, hacia las necesidades reales del mercado en cuanto a integración de datos distribuidos.

Referencias:

[Abi97] Serge Abiteboul. Querying semi-structured data. En Proceedings of the International Conference on Database Theory (ICDT). 1997

[ACPS96] S. Adali, K. Candan, Y. Papakonstantinou, V.S. Subrahmanian. Query Caching and Optimization in distributed mediator systems. En *Proceedings of the ACM SIGMOD Conference on Management of Data*. 1996

[AFT98] Laurent Amsaleg, Michael Franklin, Anthony Tomasic. Dynamic query operator scheduling for wide-area remote access. En *Distributed and Parallel Databases, vol.6 n°3*.1998

[All97] C. Allen. WIDL: Application Integration with XML. En *World Wide Web Journal, vol. 2, n°4*. 1997

[Alt] Altavista. *http://www.altavista.com*

[ALU01] Foto N. Afrati, Chen Li, Jeffrey D. Ullmann. Generating Efficient Plans for Queries Using Views. En *Proceedings of the ACM SIGMOD*. 2001

[AK97] Naven Ashish, C. Knoblock. Wrapper generation for semi-structured Internet sources. En *SIGMOD Record vol. 26 n°4*. 1997

[Aut] Autonomy Corporation. *http://www.autonomy.com*

[Bae98] Ricardo Baeza Yates, Berthier Ribeiro-Neto. Modern Information Retrieval. Addison-Wesley. 1999

[Biw] Buscador en Internet de Webs Españolas (BIWE). *http://www.biwe.es*

[BFG01] R. Baumgartner, S. Flesca, G. Gottlob. Visual Web Information Extraction with Lixto. In *Proceedings of the VLDB Conference*. 2001

[Bla96] Jose A. Blakeley. Data access for the masses through OLE DB. En *Proceedings of ACM SIGMOD Conference*. 1996

[BLP01] D. Buttler, L.Liu, C. Pu. A fully automated object extraction system for the world wide web. En *Proceedings of IEEE International Conference on Distributed Computing Systems*. 2001

[Bn] Barnes & Noble. *http://www.bn.com*

[BP98] Sergey Brin, Lawrence Page. The anatomy of a large-scale hypertextual web search engine. En *Proceedings of the International WWW Conference*. 1998

[BKLW99] Busse, S., Kutsche, R.-D., Leser, U., Weber H. Federated Information Systems: Concepts, Terminology and Architectures. Technical Report Nr. 99-9, TU Berlin.1999

[CAL00] D. Calvanese, G. De Giacomo, M. Lenzerini, M. Vardi. Answering regular-path queries using views. En *Proceedings of ICDE*. 2000

[CFRS01] D. Chamberlin, D. Florescu, J. Robie, J. Simeon, M. Stefanescu. XQuery. A query language for XML. Technical Report WWW Consortium. 2000. Disponible en *http://www.w3.org/TR/xquery/*

[Cha00] S. Chaudhuri. An Overview of Query Optimization in Relational Systems. Microsoft Corporation. 2000

[CK98] J. Carriere, R. Kazman. Webquery: Searching and visualizing the web through connectivity. En *Proceedings of the International WWW Conference*. 1998

[Coh98] William Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. En *Proceedings of ACM SIGMOD*. 1998

[DD99] Ruxandra Domenig, Klaus R. Dittrich. An Overview and Classification of Mediated Query Systems. En *SIGMOD RECORD, vol.28 nº3*. 1999

[DEW97] R. Doorembos, O. Etzioni, D. Weld. Scalable comparison-shopping agent for the World Wide Web. En *Proceedings of the International Conference on Autonomous Agents*. 1997

[DG97] Oliver M. Duschka, Michael R. Genesereth. Query planning in infomaster. En *Proceedings of the ACM Symposium on Applied Computing*. 1997

[DG97b] Oliver M. Duschka, Michael R. Genesereth. Answering recursive queries using views. En *Proceedings of PODS*. 1997

[DGL00] Oliver Duschka, Michael Genesereth, Alon Levy. Recursive query plans for data integration. En *Journal of Logic Programming, vol. 43 nº1*. 2000

[DH97] D. Dreilinger, A.E. Howe. Experiences with selecting search engines using metasearch. En *ACM Transactions on Information Systems, vol. 15, nº3*. 1997

[FHM94] Douglas Fang, Joachin Hammer, Dennis McLeod. The identification and resolution of semantic heterogeneity in multidatabase systems. En *Multidatabase Systems: An Advanced Solution for Global Information Sharing*. En *IEEE Computer Society Press*. 1994

[Eik99] Line Eikvil. Information Extraction from World Wide Web: A Survey. *Rapport* Nr. 945. 1999. ISBN 82-539-0429-0

[FLM98] Daniela Florescu, Alon Levy, Albert Mendelzon. "Database Techniques for the World-Wide Web: A Survey". En SIGMOD RECORD, vol. 27 nº3 . 1998

[FRV96] Daniela Florescu, Louiqa Raschid, Patrick Valduriez. A methodology for query reformulation in cis using semantic knowledge. En *Journal of Intelligent & Cooperative Information Systems, special issue on Formal Methods in Cooperative Information Systems, vol. 5 nº4.* 1996

[GMPQ95] H. García-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, J. Widow. The TSIMMIS approach to mediation: Data models and languages. *En Proceedings of NGITS (Next Generation Information Technologies and Systems).* 1995

[Gold99] Beth Gold-Bernstein. EAI Market Segmentation. En *Enterprise Application Integration Journal July/August 1999.*

[Goo] Google. http://www.google.com

[GRV98] J. Gruser, L. Raschid, M. Vidal and L. Bright. Wrapper Generation for Web Accesible Data Sources. En *Proceedings of CoopIS Conference*. 1998

[HD98] C-H. Hsu, M-T Dung. Generating Finite-state Transducers for semiestructured Data Extraction from the Web. En *Proceedings of the fifteenth National Conference on Artificial Intelligence (AAAI).* 1998

[HGC97] J. Hammer, H. García-Molina, J. Cho, R. Aranha, A. Crespo. Extracting semi-structured data from the web. En *Proceedings of Workshop on Management of Semi-structured data.* 1997

[HGN98] Joachim Hammer, Héctor García-Molina, Svelotzar Nestorov, Ramana Yerneni, Markus M. Breunig, Vasilis Vassalos. Template-based wrappers in the TSIMMIS system. En *Proceedings of ACM SIGMOD Conference.* 1998

[HKWY97] Laura Hass, Donald Kossmann, Edward Wimmers, Jun Yang. Optimizing queries across diverse data sources. En *Proceedings of the International Conference on Very Large Databases (VLDB).* 1997

[HS01] Justo Hidalgo, David Sánchez. Modelos de Costes para Bases de Datos Virtuales. *Technical Report DENODO Technologies.* Solicitar a jhidalgo@denodo.com

[HTM99] The W3 Consortium. HTML 4.01 Specification. *http://www.w3.org/TR/html4/*

[HTT97] The W3 Consortium. HyperText Transfer Protocol v1.1. *http://www.w3.org/Protocols/rfc2068/rfc2068*

[HTT01] Ronald Tschalar. HTTPClient v 0.3.3. http://www.innovation.ch/java/HTTPClient/

[HZ96] Richard Hull, Gang Zhou. A framework for supporting data integration using the materialized and virtual approaches. En *Proceedings on ACM SIGMOD Conference*. 1996

[IBM98] IBM Corporation. DB2 DataJoiner. Administration Guide and Application Programming. 1998

[IEX01] Microsoft Corporation. The Internet Explorer Homepage. *http://www.microsoft.com/windows/ie/default.htm*

[ILWF00] Zachary Ives, Alon Y. Levy, Daniel S. Weld, Daniela Florescu, Marc Friedman. Adaptive Query Processing for Internet Applications. En *Data Engineering Special Issue on Adaptive Query Processing*. 2000

[JAV00] SUN Microsystems. Java 2 Standard Edition Release 1.3. 2000. *http://java.sun.com/j2se/1.3/jre/*

[JAS] Netscape Corporation. Javascript Language Specification. *http://home.netscape.com/eng/javascript/*

[JDB01] SUN Microsystems. Java Data Base Connectivity. 1997-2001. http://java.sun.com/j2se/1.3/docs/guide/jdbc/spec/jdbc-spec.frame.html

[JFL01] The Jflex team. Jflex 1.3.2: The Fast Scanner generator For Java. 2001. http://www.jflex.de

[LK01] Alexander Linden, Jim Jacobs (Gartner Group). How to swim, not sink, in the information flood. *http://www.gartner.com*

[Kher00] Mandep Khera. Customer Relationship Management – Beyond the buzz. En ITToolBox CRM. 2000. *http://www.crmassist.com/documents/document.asp?i=430*

[KLMM99] C.A. Knoblock, K. Lerman, S. Minton and I. Muslea. Accurately and Reliably Extracting Data from the Web: A Machine Learning Approach. En *Bulletin of the IEEE Computer Society Technical Committee on Data Enginnering*. 1999

[KM98] T. Kistlera, H. Marais. WebL: A Programming Language for the Web. *http://www.research.digital.com/SRC/WebL/index.html*. 1998

[KMAA98] C. A. Knoblock, S. Minton, J. Ambite, N. Ashish, I. Muslea, P.J. Mody, A.G. Philpot and S. Tejada. Modeling web sources for Information Integration. En *Proceedings of the Fifteenth National Conference on Artificial Inteligence*, Madison, 1998

[KWD97] N. Kushmerick, D.S. Weld, R. Doorembos. Wrapper induction for information extraction. En *Proccedings of the fifteenth International Joint Conference on Artificial Intelligence (IJCAI)*. 1997

[KWO96] Chung T. Kwok, Daniel S. Weld. Planning to Gather Information. En Proceedings of the AAAI Thirteenth National Conference on Artificial Intelligence. 1996

[LEX75] M.E.Lesk and E. Schmidt. Lex: A Lexycal Analyzer Generator.1978. http://www.cs.utexas.edu/users/novak/lexpaper.htm

[Lev95] Alon Y. Levy, A. Mendelzon, Y. Sagiv and D. Srivastava. Answering queries using views. En *Proceedings in the Fourteenth ACM Symposium on Principles of Database Systems*. 1995

[Lev00] Alon Y. Halevy. Theory of Answering Queries Using Views. En *ACM SIGMOD Record vol. 29, n° 4*. 2000

[LPH00] Ling Liu, Carlton Pu and Wei Han. XWRAP: An XML-enabled wrapper construction system for web information sources. En *Proceedings of the 16th International Conference on Data Engineering*. 2000

[LRO96] Alon Y. Levy, Anand Rajaraman and Joann J. Ordille. Query answering algorithms for information agents. En *Proceedings of the Thirteenth National Conference on Artificial Intelligence*. 1996

[Lyc] Lycos. http://www.lycos.com

[Met] Metacrawler. http://www.metacrawler.com

[MERR98] Merryl Lynch. Move over Yahoo!. The EIP is on its way. 1998. Ver comentario en *http://www.infoworld.com/cgi-bin/displayStory.pl?/features/990125eip.htm*

[Moz01] The Mozilla Organization. *http://www.mozilla.org*

[Mul] Multibuscador e Biwe. *http://multibuscador.biwe.com*

[NGT98] Hubert Naacke, Georges Gardarin, Anthony Tomasic. Leveraging mediator cost models with heterogeneous data sources. En *Proceedings of ICDE*. 1998

[OQL] Object Data Management Group. Object Query Language (OQL). *http://www.odmg.org*

[PGH96] Yannis Papakonstantinou, Ashish Gupta, Laura Hass. Capabilities-Based Query Rewriting in Mediator Systems. En *Proceedings of the Fourth International Conference on Parallel and Distributed Information Systems*. 1996

[PDF00] Adobe Corporation. PDF Reference Version 1.3. http://partners.adobe.com/asn/developer/acrosdk/docs/PDFRef.pdf

[Per99] The Perl Porters. Perl Regular Expressions. *http://www.perldoc.com/perl5.6/pod/perlre.html*

[PGMW95] Yannis Papakonstantinou, Héctor García-Molina and Jennifer Widow. Object Exchange across Heterogeneous Data Sources. En *Proccedings of International Conference on Data Engineering (ICDE)*.1995

[PL00] Rachel Pottinger, Alon Levy. A scalable algorithm for answering queries using views. En *Proceedings of International Conference on Very Large DataBases*. 2000

[PPR96] P. Pirolli, J. Pitkow, R. Sao. Silk from a sow's ear: Extracting usable structures from the web. En *Proceedings of ACM SIGGHI Conference*. 1996

[Qua02] Quark Corporation. *http://www.quark.com*. 2002

[Ree] Reel.com. http://www.reel.com

[RMI01] SUN Mycrosystems. Java Remote Method Invocation Specification. 1997-2001. http://java.sun.com/j2se/1.3/docs/guide/rmi/spec/rmiTOC.html

[ROH00] Roth, Ozcan, L. Haas. Cost Models DO Matter: Providing Cost Information for Diverse Data Sources in a Federated System. En *Proceedings of The VLDB Conference*. 1999

[SA99] A. Sahuguet, F. Azavant. Wysiwyg Web Wrapper Factory (W4F). En *Proceedings of the WWW Conference*. 1999

[Spe97] Ellen Spertus. ParaSite: Mining structural information on the web. En *Proceedings of the International WWW Conference*. 1997

[SSL96] Netscape Corporation. The SSL Protocol Versión 3.0. 1996. http://home.netscape.com/eng/ssl3/

[STD00] Javayel Shanmugasundaram, Kristin Tufte, David J. DeWitt, Jeffrey F. Naughton, David Maier. Architecting a network query engine for producing partial results. En *Proceedings of WebDB*. 2000

[SH01] M. Stonebraker, J. Hellerstein. Content Integration for E-Business. In *Proceedings of the ACM SIGMOD Conference*. 2001

[SQL92] American National Standardas Institute. ANSI X3.135-1992, "Database Language SQL"

[TRV98] A. Tomasic, L. Raschid, P. Valduriez. Scaling access to distributed heterogeneous data sources with Disco. En *IEEE Transactions On Knowledge and Data Engineering*.1998

[UFA98] Tolga Urhan, Michael J. Franklin, Laurent Amsaleg. Cost-based query scrambling for initial delays. En *Proceedings of ACM SIGMOD*. 1998

[Ull88] Jeffrey D. Ullmann. *Principles of Database and Knowledge-Base Systems, Vol I*. Computer Science Press. 1988

[Ull97] Jeffrey D. Ullmann. Information integration using logical views. En *Proceedings of the International Conference on Database Theory (ICDT)*. 1997

[Wie92] G. Wiederhold. Mediators in the architecture of future information systems. *Computer*, 25 (3), March 1992.

[WKM00] Du Weimin, Krishnamurthy, Shan Ming-Chien. Query Optimization in a Heterogeneous DBMS. En *Proceedings of VLDB Conference*. 2000

[Wu00] Jonathan Wu. Business Intelligence: The Transition of Data into Wisdom. En *DM Review*. 2000.
http://www.dmreview.com/portal_ros.cfm?NavID=91&EdID=2524&PortalID=17

[XML00] W3 Consortium. Extensible Markup Language (XML) 1.0 (Second Edition). 2000. *http://www.w3.org/TR/2000/REC-xml-20001006*

[XMS01] W3 Consortium. XML Schema Specification 1.0. 2001. http://www.w3.org/XML/Schema

[YLGU99] Ramana Yerneni, Chen Li, Héctor García-Molina, Jeffrey Ullmann. Computing Capabilities of Mediators. En *Proccedings of the ACM SIGMOD Conference*. 1999

[YPAG98] Ramana Yerneni, Yannis Papakonstantinou, Serge Abiteboul, Héctor García-Molina. Fusion queries over internet databases. En *Proceedings of the Conference on Extending Database Technology (EDBT)*. 1998