Definición del modelo y esquema del Almacén de Datos en función de las características temporales de los sistemas operacionales componentes

Dirigida por

Dr. José Samos Jiménez
Dra. Cecilia Delgado Negrete

Francisco Araque Cuenca
Granada, Noviembre del 2005

E. T. S. de Ingeniería Informática
Dpto. Lenguajes y Sistemas Informáticos

ugr Universidad de Granada

Editor: Editorial de la Universidad de Granada
Autor: Francisco Araque Cuenca
D.L.: Gr. 2316 - 2005
ISBN: 84-338-3732-x

Definición del modelo y esquema del Almacén de Datos en función de las características temporales de los sistemas operacionales componentes

Dr. José Samos Jiménez
Director

Dra. Cecilia Delgado Negrete
Directora

Francisco Araque Cuenca
Doctorando

Granada, Noviembre del 2005

ugr Universidad de Granada
Dpto. Lenguajes y Sistemas Informáticos
E.T.S. de Ingeniería Informática

Definición del modelo y esquema del Almacén de Datos en función de las características temporales de los sistemas operacionales componentes

Dr. José Samos Jiménez
Director

Dra. Cecilia Delgado Negrete
Directora

Francisco Araque Cuenca
Doctorando

Granada, Noviembre del 2005

E. T. S. de Ingeniería Informática
Dpto. Lenguajes y Sistemas Informáticos

Agradecimientos

En primer lugar, quiero hacer llegar mi más sincero y profundo agradecimiento al doctor José Samos y a la doctora Cecilia Delgado por aceptar la dirección de esta tesis, así como por su inestimable ayuda, y el ejemplo profesional y personal que me ha demostrado durante estos años.

También quiero mostrar mi gratitud a los compañeros del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada, y especialmente a los miembros del Grupo de Investigación en Sistemas de Gestión de Bases de Datos Federadas, Eladio Garví, Emilia Ruíz y Alberto Salguero, por sus valiosos comentarios.

A la ayuda facilitada por los proyectos del programa CICYT TIC1999-1078-C02-02 y TIC 2000-1723-C02-02.

Mi agradecimiento también al Departamento de Informática de la Universidad de Jaén, por facilitarme, en todo momento, mi labor.

Por último, y no por ello menos importante, a mi familia, ellos saben las razones.

Resumen

Un *almacén de datos* contiene una copia de los datos de los sistemas operacionales, y ofrece una visión histórica de los datos con una estructura especialmente adecuada para realizar consultas y análisis

El diseño del esquema de un almacén de datos debe ser orientado al almacenamiento en sí, sin tener en cuenta las posibles consultas que se puedan llegar a realizar sobre él. Las consultas se realizarán principalmente sobre BD multidimensionales diseñadas con este propósito y cuyos datos se obtienen a partir del almacén. Llegado a este punto, es muy importante tener en cuenta la dimensión temporal, vital en cualquier tarea de análisis. El esquema del almacén de datos ha de ser capaz de reflejar las características temporales de los datos. De la misma manera, será igualmente importante estudiar los mecanismos de extracción, de este tipo de datos, de los sistemas operacionales componentes.

Por otra parte, también es necesario trasladar los datos de los sistemas operacionales al modelo de datos del almacén, así como comprobar si los datos de las fuentes de datos han cambiado desde el último acceso para, en su caso, actualizar el almacén de datos. Para realizar este proceso de integración es adecuado usar un modelo orientado a objetos (O-O) como modelo canónico de datos. El estándar ODMG define un conjunto de especificaciones que establece cómo se deben definir y consultar bases de datos orientadas a objetos. Actualmente, dicho estándar no ha sido reconocido como tal por ningún comité de estandarización, aunque hoy en día es el estándar de hecho. Sin embargo, dicho estándar no incluye elementos temporales. La inclusión de esta funcionalidad permitiría representar elementos y propiedades temporales en los esquemas tanto de las fuentes de datos como del propio almacén de datos.

Las bases de datos de tiempo real y las bases de datos temporales utilizan conceptos que pueden ser aplicados en la tarea de enriquecimiento del modelo O-O para facilitar el proceso de definición del modelo y la actualización del almacén. Tener en cuenta las características temporales de las fuentes de datos, podrá ayudar a la hora del diseño del esquema del almacén de datos, ya que el diseñador podrá disponer de información no sólo de los datos que las diferentes fuentes pueden ofrecer, sino también de sus características temporales, siendo ambos componentes del esquema del almacén de datos.

La integración de fuentes de datos para su incorporación al almacén de datos es un campo que ha sido estudiado desde el punto de vista de resolver diferentes problemas, tales como: Integración de las propiedades semánticas de los datos, integración de niveles de seguridad, etc. Sin embargo, hasta ahora no se han tenido en cuenta las propiedades temporales de los datos. Estas propiedades se pueden deducir tanto de la propia fuente en la que se encuentra el dato, como del método de extracción utilizado para la obtención del mismo. La integración de las propiedades temporales de los datos implica disponer de algoritmos que nos resuelvan que, si datos procedentes de diferentes fuentes y teniendo en cuenta las propiedades temporales de los mismos, es posible su integración para ser incorporados al almacén.

En esta tesis se estudia el problema de definición del modelo y esquema del almacén de datos en función de las características temporales de los sistemas operacionales componentes. Partiendo del modelo de objetos ODMG como modelo canónico de datos, se propone un enriquecimiento del mismo para incorporar elementos temporales y se extienden los metadatos para contemplar los elementos añadidos al modelo. Se proponen también algoritmos para la integración de las propiedades temporales de los datos antes de ser incorporados al almacén de datos. Por último, también se aportan soluciones al problema de la actualización del almacén; estas soluciones se basan en las propiedades temporales de los datos y en los requerimientos del administrador del propio almacén.

Indice general

Indice de figuras

Indice de tablas

Capítulo 1

INTRODUCCIÓN

1.1 Descripción del problema

Un *almacén de datos* (AD) [Inmo92] es una base de datos (BD) que contiene una copia de los datos de los sistemas operacionales (fuentes de datos) con una estructura especialmente adecuada para realizar consultas y análisis (ofrece una visión histórica de los datos de los sistemas operacionales). El ámbito es uno de los elementos característicos de los almacenes de datos: Es toda la empresa. Se usa el término *data mart* para designar a los almacenes de datos cuyo ámbito es más reducido, normalmente un departamento o área específica dentro de la empresa.

Los datos procedentes de las aplicaciones, sistemas operacionales componentes y otras fuentes de información, se integran en el *almacén de datos operacionales* (ADO, "Operational Data Store"), que es una colección que contiene datos detallados para satisfacer las necesidades operacionales de acceso colectivo e integrado a los datos de la corporación a la que sirve. Generalmente, a partir del ADO se define el *almacén de datos corporativo*, aunque también existe la posibilidad de obtener los datos a cargar en el almacén directamente de los sistemas operacionales componentes. Los data marts se definen siempre a partir del almacén de datos.

Es muy frecuente encontrar asociado el concepto de almacén de datos con su implementación sobre BD relacionales o bien multidimensionales (construidas sobre BD relacionales o bien con estrategias específicas de almacenamiento multidimensional de los datos). Sin embargo, el uso de las bases de datos orientadas a objetos (BDOO) puede resultar de gran utilidad para el modelado e implementación de los almacenes de datos.

El diseño del esquema de un almacén de datos debe ser orientado al almacenamiento en sí, sin tener en cuenta las posibles consultas que se puedan llegar a realizar sobre él. Las consultas se realizarán principalmente sobre BD multidimensionales diseñadas con este propósito y cuyos datos se obtienen a partir del almacén. Llegado a este punto, es muy

importante tener en cuenta la dimensión temporal, vital en cualquier tarea de análisis. El esquema del almacén de datos ha de ser capaz de reflejar las características temporales de los datos. De la misma manera, será igualmente importante estudiar los mecanismos de extracción, de este tipo de datos, de los sistemas operacionales componentes (fuentes de datos).

Por otra parte, también es necesario trasladar los datos de las fuentes al modelo de datos del almacén, así como comprobar si los datos de las fuentes de datos han cambiado desde el último acceso para, en su caso, actualizar el almacén de datos. Para realizar este proceso de integración es adecuado usar un modelo orientado a objetos (O-O) como modelo canónico de datos, tal como se desprende de los resultados presentados en [Saltor et al., 91]. El estándar ODMG define un conjunto de especificaciones que establece cómo se deben definir y consultar bases de datos orientadas a objetos. Actualmente, dicho estándar no ha sido reconocido como tal por ningún comité de estandarización, aunque hoy en día es el estándar de hecho. Sin embargo, dicho estándar no incluye elementos temporales. La inclusión de esta funcionalidad permitiría representar elementos y propiedades temporales en los esquemas tanto de las fuentes de datos como del propio almacén de datos.

Las bases de datos de tiempo real y las bases de datos temporales utilizan conceptos que pueden ser aplicados en la tarea de enriquecimiento del modelo O-O para facilitar el proceso de definición del modelo y la actualización del almacén. En la mayoría de los trabajos actuales, la extracción de datos de las fuentes se realiza según los requisitos específicos del problema planteado, con módulos desarrollados ad hoc para llevar a cabo dicha extracción. Tener en cuenta las características temporales de las fuentes de datos, podrá ayudar a la hora del diseño del esquema del almacén de datos, ya que el diseñador podrá disponer de información no sólo de los datos que las diferentes fuentes pueden ofrecer, sino también de sus características temporales, siendo ambos componentes del esquema del almacén de datos.

La integración de fuentes de datos para su incorporación al almacén de datos es un campo que ha sido estudiado desde el punto de vista de resolver diferentes problemas, tales como: Integración de las propiedades semánticas de los datos, integración de niveles de seguridad, etc. Sin embargo, hasta ahora no se han tenido en cuenta las propiedades temporales de los datos. Estas propiedades se pueden deducir tanto de la propia fuente en la que se encuentra el dato, como del método de extracción utilizado para la obtención del mismo. La integración de las propiedades temporales de los datos implica disponer de algoritmos que nos resuelvan que, si datos procedentes de diferentes fuentes y teniendo en cuenta las propiedades temporales de los mismos, es posible su integración para ser incorporados al almacén.

En esta tesis se estudia el problema de definición del modelo y esquema del almacén de datos en función de las características temporales de los sistemas operacionales componentes. Partiendo del modelo de objetos ODMG como modelo canónico de

datos, se propone un enriquecimiento del mismo para incorporar elementos temporales y se extienden los metadatos para contemplar los elementos añadidos al modelo. Se proponen también algoritmos para la integración de las propiedades temporales de los datos antes de ser incorporados al almacén de datos. Por último, también se aportan soluciones al problema de la actualización del almacén; estas soluciones se basan en las propiedades temporales de los datos y en los requerimientos del administrador del propio almacén.

1.2 Contexto y motivación

Esta tesis se ha desarrollado en el contexto de los proyectos CICYT "Sistema cooperativo para la integración de fuentes heterogéneas de información y almacenes de datos" (TIC 1999-1078-C02-02), y "Acoplamientos fuerte y débil: Integración de fuentes de datos estructurados y semiestructurados" (TIC 2000-1723-C02-02). En dichos proyectos han participado investigadores de las universidades Politécnica de Cataluña, Granada, Lérida y Almería. El objetivo principal de ambos proyectos consistía en el desarrollo de sistemas que permitan la integración de fuentes heterogéneas de información y almacenes de datos, ya sean los datos estructurados o semiestructurados; se trataba de desarrollar una capa software que permita el acoplamiento de los sistemas heterogéneos ofreciendo un acceso integrado a los datos.

1.3 Solución propuesta

Este trabajo tiene dos objetivos: El primero, proponer un modelo O-O enriquecido con características temporales para la definición del esquema del almacén de datos y el esquema de las fuentes que le proporcionan los datos al almacén; con este modelo asimismo se pretende estudiar el proceso de definición de los distintos esquemas en base a las características temporales de los datos. El segundo objetivo es proponer algoritmos para la integración de las propiedades temporales de los datos y del propio almacén.

Pasos a realizar:

- Enriquecimiento de un modelo de datos O-O con elementos temporales.

  La propia definición del almacén de datos pone de manifiesto la importancia de la dimensión temporal en este tipo de bases de datos. Dado que el modelo de datos canónico es O-O y el del almacén de datos también, es necesario estudiar la representación de elementos temporales en este tipo de modelos de datos. La representación diferenciada del tiempo, dentro del esquema del almacén, facilitará la comprensión y manejo de sus contenidos.

- Definición del esquema del almacén de datos.

  Una vez que se disponga de la representación adecuada para los elementos temporales del almacén, habrá que establecer los mecanismos de definición de su

esquema. La información que puede contener siempre estará condicionada por los datos disponibles en las fuentes de datos. Por lo tanto, es necesario determinar qué información se puede extraer de esas fuentes de datos y cómo quedará representada en el almacén.

- Definición del esquema de las fuentes de datos

  En el almacén de datos la extracción de información hay que hacerla por adelantado, sin esperar las consultas de los usuarios. De esta manera, se hace necesario un estudio de la representación de los elementos (basados en conceptos de BD de tiempo real y BD temporales) que permitan expresar las características de obtención de los datos de las distintas fuentes de información.

- Integración de las propiedades temporales de los datos

  El almacén de datos se alimenta de fuentes de datos heterogéneas. Para cada tipo de fuente se utiliza un método de extracción que viene determinado por las propiedades de dicha fuente. Es necesario disponer de mecanismos que ayuden al administrador del almacén, en la fase de integración, a decidir si los datos pueden ser integrados o no.

- Actualización del almacén de datos

  El problema de la actualización del almacén de datos, a partir de las fuentes de datos, es uno de los más importantes en su construcción. No es suficiente extraer los datos, sino que también es necesaria la actualización del almacén.

1.4 Objetivos de la tesis

Los principales objetivos que se pretenden alcanzar son:

- Estudiar y clasificar los diferentes métodos de extracción de datos de las fuentes existentes.
- Estudiar el estándar ODMG: modelo de objetos, metadatos y lenguaje de definición de objetos.
- Proponer propiedades temporales para los métodos de extracción de datos.
- Proponer una arquitectura funcional para almacenes de datos.
- Proponer una arquitectura de operación para almacenes de datos.
- Enriquecer el modelo de objetos ODMG con elementos temporales.
- Extender los metadatos de ODMG para incorporar elementos temporales.
- Desarrollar algoritmos para la integración de las propiedades temporales de los datos.
- Desarrollar algoritmos para la actualización del almacén de datos.

1.5 Estructura de la tesis

La tesis se estructura en 8 capítulos, incluido éste de Introducción:

Marco de referencia

En el Capítulo 2 se estudia el problema de la integración en general haciendo hace más énfasis en los problemas relacionados con el tiempo. Se han propuesto muchas arquitecturas para sistemas de bases de datos federadas con el propósito de proporcionar acceso integrado a las fuentes, nosotros partimos de la propuesta por Sheth y Larson que es ampliada para contemplar los parámetros de tiempo necesarios. En este capítulo se explica dicha arquitectura y las modificaciones necesarias en los diversos niveles para poder definir esquemas componentes y esquemas de exportación temporales. Dado que necesitaremos un modelo canónico de datos orientado a objetos, se presentan las principales propiedades que éste debe tener así como las principales características del modelo de datos corporativo. También se estudian y clasifican los diferentes métodos de extracción de datos.

El estándar ODMG 3.0

En el Capítulo 3 se revisan algunos modelos temporales, principalmente aquellos que han partido de ODMG para realizar su propuesta de extensión con elementos temporales. También se comentan los principales conceptos de interés para esta tesis y que son utilizados en otras áreas tales como bases de datos de tiempo real y bases de datos temporales. Teniendo en cuenta los modelos temporales existentes y los conceptos que necesitamos para nuestro trabajo, se propone el modelo de tiempo que será utilizado en el resto de la tesis. También en este capítulo se resumen las especificaciones del estándar ODMG 3.0. De forma breve se realiza un repaso de los aspectos más importantes de este estándar, concretamente el modelo de objetos, los metadatos relacionados con la definición de esquemas y el lenguaje de definición de objetos ODL (*Object Definition Language*). Asimismo, en este capítulo se citan los distintos miembros de ODMG y se resumen las características de los principales productos existentes en el mercado relacionados con el almacenamiento de objetos persistentes, con el objetivo de ver en qué medida implementan este estándar.

Enriquecimiento de ODMG con elementos temporales

En el Capítulo 4 se amplia el modelo de objetos de ODMG con los elementos temporales necesarios para poder definir el esquema de las fuentes de datos y del almacén de datos. Se extiende la jerarquía de tipos de ODMG con nuevos tipos con semántica temporal y se incorporan nuevos elementos al modelo para representar objetos con propiedades temporales. En este capítulo también se modifica el ODL de ODMG para que refleje las extensiones propuestas.

Extensión de los metadatos de ODMG para contemplar elementos temporales

En el Capítulo 5 se propone una extensión de los metadatos de ODMG para que contemple los elementos temporales propuestos en el Capítulo 3. También se extienden los metadatos para incorporar la definición de las fuentes de datos que forman parte del almacén y se propone una nueva relación. Concretamente, se propone soporte para la definición de atributos, clases e interfaces temporales, y se introducen las modificaciones necesarias para expresar la relación entre el almacén de datos y las fuentes de datos.

Algoritmos de integración de propiedades temporales de los datos y arquitectura de integración

En el Capítulo 6 se proponen las propiedades temporales de los datos y se definen los parámetros temporales de cada uno de los métodos de extracción. Una vez fijados dichos parámetros temporales se estudia la integración temporal primero a nivel abstracto y luego proponiendo algoritmos que se encargarán de integrar las propiedades temporales de los datos según el método de extracción utilizado para obtenerlos. En este capítulo también se propone una arquitectura funcional de integración temporal que consta de dos fases: La primera es el procesador temporal que se encargaría de que se satisfagan ciertos parámetros temporales comunes a cualquier tipo de fuente para que la integración pueda llevarse a cabo. La segunda es el generador de metadatos de refresco, que determina los parámetros más adecuados para llevar a cabo el refresco de los datos.

Operación del sistema

En el Capítulo 7 se estudia el problema del refresco del almacén de datos y se propone una arquitectura de operación para realizar el refresco de los datos según el método de extracción utilizado y las propiedades temporales. El refresco consta de dos fases incluidas en la arquitectura y controladas por dos módulos: El gestor de refresco que se encarga de garantizar que las fuentes de datos son accedidas de manera coherente cada vez que se realiza un refresco de los datos, y el gestor de consultas que se encarga de transformar las consultas entre los diferentes esquemas de datos del sistema. Para finalizar el capítulo se proponen algoritmos de refresco para cada método de extracción.

Conclusiones

En el Capítulo 8 se presentan las conclusiones de de esta tesis y se presenta el trabajo futuro.

Capítulo 2

MARCO DE REFERENCIA

La integración de información para el acceso integrado a los datos es una tarea compleja debido a que las fuentes tienen interfaces, funcionalidades y estructura diferentes lo que conlleva problemas de compatibilidad. Dado el gran número de fuentes disponibles es posible que encontremos datos que estén repetidos en diferentes fuentes lo que hace que la integración sea más costosa, difícil de automatizar y que consume un tiempo considerable. En el caso de los almacenes de datos la integración incluye más factores que la hacen aún más compleja. Uno de los factores que hay que tener en cuenta en la integración es el tiempo, en concreto las propiedades temporales de los datos. Estas vendrán determinadas por las fuentes de datos donde se encuentra el dato y por lo métodos de extracción elegidos para obtener el dato. En este capítulo se introducen los conceptos necesarios y se propone una arquitectura de referencia que será utilizado durante el resto de la tesis.

El capítulo se estructura como sigue. En la sección 2.1 se introduce el problema de la integración de información. En la sección 2.2 se presenta el problema de la integración de información en almacenes de datos y se revisan algunas propuestas existentes. En la sección 2.3 se comenta el problema de la integración pero centrándose en el factor temporal. En la sección 2.4 se presenta nuestra arquitectura de partida y la arquitectura propuesta para abordar los problemas de integración temporal. En la sección 2.5 se estudia como se construye el sistema federado. En la sección 2.6 se presenta la arquitectura propuesta de manera detallada explicando brevemente cada módulo. En la sección 2.7 comentamos las características deseables del modelo de datos corporativo del almacén de datos. En la sección 2.8 se presentan los métodos de extracción y las fuentes de datos y se propone una clasificación de los mismos. Por último, la sección 2.9 resume las conclusiones de este capítulo.

2.1 Introducción

Los datos de interés para una empresa o un organismo público se encuentran, cada vez con mayor frecuencia, desperdigados en múltiples fuentes. Hasta no hace mucho, estos datos residían en ficheros y bases de datos, y su estructura venía dada por el *esquema*; eran *datos estructurados*. En estos últimos tiempos, sin embargo, la aparición de Internet y la proliferación de páginas en la *World Wide Web* ha dado lugar a información que no sigue otra estructura que la de las marcas de un *lenguaje de marcas* (*Markup Language*), como el *HTML* y el *XML*; son *datos semiestructurados*.

Se han propuesto numerosos trabajos para el acceso a los datos almacenados en bases de datos heterogéneas y otras fuentes de datos tales como sistemas legados [Shet90], [Salt96]. Con el crecimiento y perfeccionamiento de las redes en general y de internet en particular el número de fuentes de las que podemos extraer datos se ha incrementado notablemente. Aunque esto último es una ventaja también plantea numerosos problemas. Internet es un entorno dinámico y no siempre se puede asegurar que una determinada fuente esté disponible. Además estas fuentes pueden, y de hecho tienen, interfaces y funcionalidades diferentes lo que conlleva problemas de compatibilidad. Dado el gran número de fuentes disponibles es posible que encontremos datos que estén repetidos en diferentes fuentes. Aunque esto puede parecer una ventaja, dado que podemos disponer de más fuentes donde encontrar lo que buscamos, en la práctica significa que los planes de ejecución de una consulta son más costosos y complejos. En este punto un tema de trabajo importante es la integración de información proveniente de diferentes fuentes [Levy95], [Zach00], [Doan00], [Berns00], siendo esta una tarea difícil de automatizar y que consume un tiempo considerable. En numerosas ocasiones hay que llegar a un compromiso entre la completa integración de la información y la necesidad de obtener el resultado de una consulta de una forma rápida aunque la información obtenida no sea del todo completa.

Para que un usuario pueda acceder a múltiples fuentes de datos de un modo *integrado*, es decir, como si se tratase de una sola base de datos, hace falta instalar un sistema que produzca un *acoplamiento* entre esas fuentes. Sin acoplamiento, el acceso no es integrado, y hace falta acceder separadamente a cada una de las fuentes (para lo cual hay que conocer qué datos contiene, en qué modelo/formato, y uno de sus lenguajes de acceso), y luego combinar las respectivas respuestas (pasándolas a un formato común y eliminando redundancias).

Cuando los datos de todas las fuentes son estructurados, y es posible llevar un control sobre ellas, el acoplamiento puede ser *fuerte*: el sistema que realiza el acoplamiento se llama federado, y existe un administrador federado que resuelve las heterogeneidades entre las fuentes y prepara esquemas externos para los usuarios.

Si hay fuentes con datos semiestructurados, o no es posible ningún control sobre algunas fuentes, ya que son totalmente autónomas, como ocurre en el caso de Internet y las

páginas Web, el acoplamiento debe ser *débil*: no existe la figura de administrador federado, y cada usuario ha de solucionar las heterogeneidades y construir sus esquemas.

Tanto en el caso de acoplamiento fuerte como en el del débil, este acceso integrado puede hacerse:

- A través de una consulta que accede directamente a las fuentes preexistentes, que interoperan formando un *sistema cooperativo*.
- Consultando un *almacén de datos (Data Warehouse)*, en el que se vuelcan y consolidan los datos de esas fuentes. Los datos están materializados y son gestionados por el sistema cooperativo.
- Mediante una consulta que puede acceder tanto a las fuentes preexistentes como a datos materializados vistos de forma uniforme mediante un esquema común y que están bajo el control del sistema cooperativo.

Existen propuestas de arquitecturas basadas en componentes los cuales se encargan de resolver los problemas antes citados [Grus98], [Cres01], [Garc98], [Levy96], [Chen00]. Estos componentes, se encargarán entre otras cuestiones, de extraer los datos de las fuentes, traducir las consultas realizadas entre los diferentes modelos de datos utilizados, detectar cambios en las fuentes y resolver las heterogeneidades semánticas.

2.2 Integración de Información y Almacén de datos

El problema de la integración de información[1] surge de la necesidad de, por una parte aunar los datos dispersos a lo largo de diversas fuentes y, por otra parte, por la necesidad de dotar de una representación homogénea de los datos recopilados. Los sistemas de información actuales involucran a un gran número de fuentes de datos a menudo distribuidas geográficamente. Un sistema de información debería disponer de un interface para que los usuarios tuvieran la visión de que están consultando o accediendo a un conjunto de datos homogéneos, independientemente de cómo el sistema de información internamente haya obtenido e integrado esos datos provenientes de las fuentes de datos [Levy95]. Para resolver estas heterogeneidades semánticas necesitamos conocer explícitamente la estructura de las fuentes.

A continuación se presentarán los tipos de integración y cómo el almacén de datos enfoca el problema de la integración.

2.2.1 Integración

El problema de la integración ha sido abordado en varias áreas de investigación [Calv98]:

[1] *Datos*: colección de observaciones adquiridas en varios puntos a lo largo del proceso de negocio. *Información*: la utilización de los datos para tomar decisiones. Utilizaremos indistintamente el término dato y el término información.

- Bases de datos: se estudia la integración de los esquemas y de los datos.
- Sistemas de Información Cooperativos: en el que varios componentes cooperan para realizar una determinada tarea, por ejemplo compartir datos.
- Sistemas de Información Globales: en este caso fuentes de datos heterogéneas se consultan de forma integrada, como si se tratara de una sola fuente de datos. Por ejemplo varios sitios Web a los que se accede como si de uno solo se tratara.
- Representación del conocimiento: la integración se considera de una forma general. Se estudian técnicas para unir diferentes teorías u otra clase de conocimiento expresado en una forma más general de la que ofrecen sólo los datos.

Otro punto a considerar es el contexto donde se realiza la integración. En este caso hay que tener en cuenta el tipo de información de entrada y de salida del proceso de integración así como el objetivo del mismo. Podemos distinguir entre:

- Integración de esquemas. En este caso la entrada es un conjunto de esquemas de datos (de las fuentes) y la salida es un solo esquema global que representa la integración de todos los esquemas de entrada. La salida incluye también la especificación de cómo trasladar cada esquema de datos de las fuentes al esquema global. Este tipo de integración es el más utilizado en el campo de bases de datos federadas.
- Integración de los datos virtuales. La entrada es un conjunto de fuentes de datos y la salida es una especificación de cómo proporcionar un acceso global e integrado a las fuentes para satisfacer ciertas necesidades de información. Se mantiene la autonomía de las fuentes. Se utiliza como respuesta a consultas en bases de datos federadas. Este tipo de integración es el más utilizado en bases de datos.
- Integración de los datos materializados. La entrada es un conjunto de fuentes de datos y la salida es un conjunto de datos que representan una visión integrada de los datos de entrada. Este tipo de integración es utilizado fundamentalmente en el almacén de datos.

En los dos primeros tipos de integración (de esquemas y de datos virtuales) el enfoque para acceder a los datos es *bajo-demanda* en el cual a petición de una consulta realizada por un usuario del sistema se accede a los datos, se integran y se devuelve el resultado al usuario. En este caso los datos no están materializados.

2.2.2 Almacén de Datos

A parte del enfoque de acceder a la información bajo-demanda, existe el enfoque del *Almacén de Datos (Data Warehouse)*. Un *almacén de datos* [Inmo92] es una BD que contiene una copia de datos de los sistemas operacionales (fuentes de datos) con una estructura especialmente adecuada para realizar consultas y análisis (ofrece una visión histórica de los

datos de los sistemas operacionales). El ámbito es uno de los elementos característicos de los almacenes de datos: su ámbito es toda la empresa, en este caso lo llamaremos *almacén de datos*. Se usa el término *data mart* para designar a los almacenes de datos cuyo ámbito es más reducido, normalmente un departamento o área específica dentro de la empresa.

Entre las características que diferencian al almacén de datos de otros enfoques encontramos las siguientes [Rund00]:

- Los datos se extraen de los sistemas operacionales, se transforman según sea necesario, se integran con datos provenientes de otros sistemas operacionales y se almacenan en el almacén de datos.
- Cuando se realiza una consulta, ésta se ejecuta en el almacén de datos, sin que sea necesario acceder a las fuentes de datos.
- Cuando se produce un cambio de los datos en los sistemas operacionales y éste se considera de relevancia, se propagan los cambios al almacén de datos y se actualiza.

En la Figura 1 se presenta un resumen de la arquitectura de construcción de almacenes de datos propuesta en [Inmo98]. Los datos procedentes de las aplicaciones, BDs componentes y otras fuentes de datos son integrados directamente en el almacén de datos. Los data marts son definidos a partir del data warehouse.

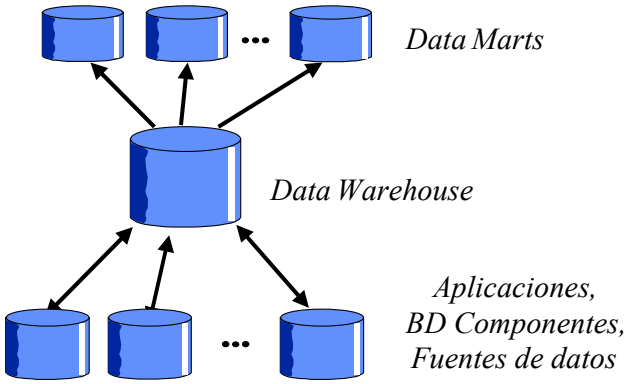

Figura 1. Arquitectura de construcción de almacenes de datos.

En la Figura 2 podemos ver la implementación de la arquitectura que proponen [Wido95] y [Garc98] y que muestra los componentes que interactúan para extraer e integrar los datos. Se han introducido dos nuevos componentes: los *wrappers/monitors* y el *integrator*. Conectado a cada fuente de datos hay un wrapper/monitor. El wrapper se encarga de trasladar los datos del formato nativo de la fuente al formato y modelo de datos utilizado por el almacén de datos. El monitor se encarga de detectar automáticamente cambios en la fuente e informar de éstos al integrator. Cuando una nueva fuente entra a formar parte del almacén de datos o cuando se detectan cambios de interés en las fuentes, los nuevos datos o los cambios producidos en los datos se propagan al integrator. Este componente

se encarga de filtrar los datos, resumirlos, y si fuera necesario integrarlos con los datos procedentes de otras fuentes.

En la Figura 2 se muestra un almacén de datos simple y centralizado, pero éste puede ser implantado cómo una BD distribuida y se pueden utilizar técnicas de paralización para alcanzar mayores grados de rendimiento [Gatz99].

Otro tipo de datos son los metadatos que contienen información acerca de los datos [Stau99]. Es un directorio que contiene información sobre como son los datos almacenados en el almacén de datos y dónde se pueden encontrar esos datos.

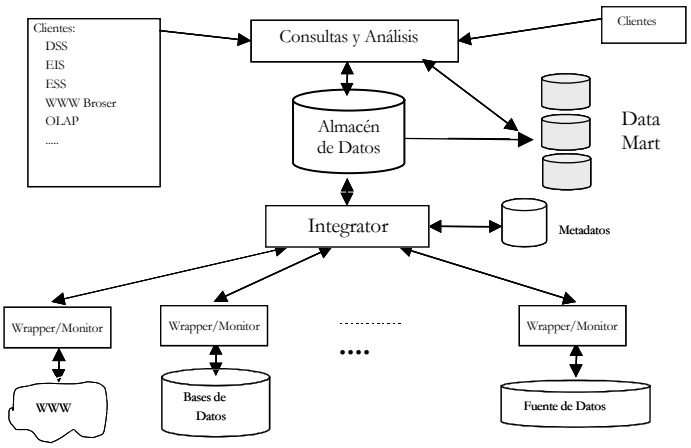

Figura 2. Arquitectura general de un Data Warehouse

La Figura 3 ilustra la arquitectura genérica de un almacén de datos. Las fuentes de datos incluyen bases de datos operacionales, ficheros flat (por ejemplo, hojas de cálculo o ficheros de textos) y bases de datos externas. Los datos se extraen de las fuentes y se cargan en el almacén de datos utilizando herramientas ETL (Extraction, Transformation and Load) [Araq02] [Araq03a]. El almacén de datos se utiliza para alimentar data marts temáticos y servidores OLAP. Los data marts son subconjuntos del almacén de datos categorizados de acuerdo a las áreas funcionales y dependiendo del dominio del problema. Los servidores OLAP son herramientas software que ayudan al usuario a preparar datos para el análisis, búsquedas, informes y data mining. El almacén de datos completo forma un sistema integrado que puede dar soporte a varios requerimientos de análisis e informes para la ayuda en la toma de decisiones [Chau97].

Por tanto, un almacén de datos junto con OLAP, capacita a los decisores para analizar y entender los problemas. OLAP analiza los datos utilizando esquemas especiales, y permite al usuario visualizar los datos usando cualquier combinación de variables. Los sistemas de almacén de datos transforman los datos operacionales en información estratégica para la toma de decisiones. Almacenan información sumarizada variante en el tiempo en lugar de datos operacionales, proporcionando respuestas a preguntas en el ámbito decisional. Para

extraer esta información de un entorno distribuido, necesitamos consultar múltiples fuentes de datos e integrar la información antes de presentar las respuestas al usuario. En el entorno de los almacenes de datos, esas consultas encuentran sus respuestas en un lugar central, de tal forma que se reduce el coste de procesamiento y manejo. Después de la carga inicial el almacén debe ser regularmente refrescado y las modificaciones de los datos operacionales producidos desde el último refresco se deben propagar al almacén, de tal forma que éste refleje de la forma más fiable posible el estado de los sistemas operacionales [Araq03] [Araq03b].

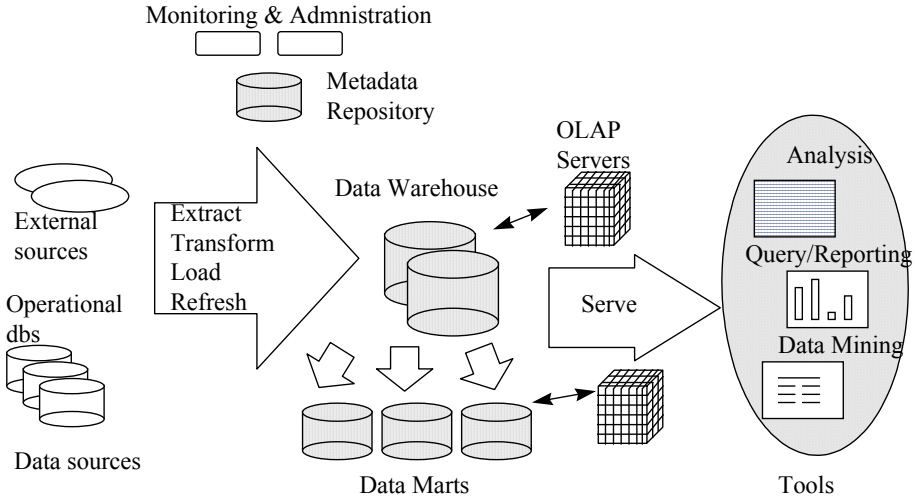

Figura 3. Arquitectura genérica de DW (Chaudhuri & Dayal, 1997).

Aunque se reconocen importantes elementos en común entre los almacenes de datos y otras áreas previamente estudiadas, no se aprovechan en su totalidad los avances conseguidos en estas áreas para su aplicación a la construcción de almacenes de datos (data warehouses y data marts), en concreto, nos referimos especialmente a las BD federadas, BD temporales, así como a la definición de esquemas externos. Situando a los almacenes de datos en el contexto adecuado se puede subsanar esta situación, éste es el objetivo de nuestro trabajo.

2.2.3 Tipos de integración

2.2.3.1 Integración de esquemas

Es la actividad de integrar varios esquemas de diferentes fuentes de datos con el objetivo de producir una descripción homogénea de los datos de interés. Tradicionalmente se llevaba a cabo en un solo paso y se obtenía un esquema global en el cual todos los datos estaban representados uniformemente. Con la disponibilidad de mayor número de fuentes dinámicas y autónomas se aborda el problema de una forma incremental. El resultado del la integración debería ser en teoría un esquema global como si se realizara en

un solo paso. En la práctica nunca se llega a alcanzar este objetivo debido a la naturaleza cambiante de las fuentes de datos.

Una forma de integración de esquemas consiste en *comparar diferentes* esquemas para determinar relaciones entre conceptos de diferentes esquemas y detectar posibles conflictos. Estos pueden ser: *conflictos de heterogeneidad*, surgen cuando se usan diferentes modelos de datos para los esquemas de las fuentes; c*onflictos de nombrado*, cuando diferentes esquemas se refieren al mismo dato usando diferente terminología; *conflictos semánticos*, debidos a la utilización de diferentes niveles de abstracción al modelar entidades del mundo real que son similares y *conflictos estructurales* que se producen cuando se utilizan diferentes estructuras para representar los mismos conceptos.

Respecto a la integración de esquemas podemos encontrar una propuesta en [Garc95]. Si las fuentes están expresadas en distintos modelos de datos una estrategia de integración es realizar un *enriquecimiento semántico* que normalmente se lleva a cabo trasladando los esquemas de las fuentes a un modelo de datos más rico, que permita representar información sobre dependencias, valores nulos y otras propiedades semánticas que permiten aumentar las posibilidades de representar los datos de las distintas fuentes. Esta metodología se explica con más detalle en la sección 2.5 y será ampliada en esta tesis para incorporar los módulos necesarios para la integración temporal.

2.2.3.2 Integración de los datos virtuales

Uno de los aspectos que distingue la integración de datos de la integración de esquemas, es el *emparejamiento de objetos,* que establece cuándo diferentes objetos en diferentes fuentes representan el mismo elemento en el mundo real, y por tanto, debería ser identificado de la misma manera. El criterio más simple para realizar esto es el basado en las llaves, que consiste en identificar los objetos que tienen la misma llave.

En cuanto a la integración de los datos virtuales comentaremos las siguientes propuestas: el Sistema Carnot, el Sistema SIMS, Sistema Information Manifold, el Proyecto TSIMMIS y por último un enfoque basado en ontologías.

En el sistema *Carnot*, descrito en [Huhs93], los esquemas individuales son trasladados al esquema global basándose en una ontología que es proporcionada por una base de conocimiento llamada *Cic*. Esta ontología está expresada en un lenguaje de primer orden llamado *lenguaje de contexto global* (LCG). La alta expresividad de este lenguaje, permite representar en el esquema global, los metamodelos de varios formalismos de esquemas y todo el conocimiento disponible sobre esquemas individuales, incluyendo restricciones de integridad y operaciones permitidas. Cuando se ha concluido el mapeo de esquemas, las consultas y actualizaciones son realizadas traduciéndolas primero al LCG y después distribuyéndolas a los diferentes subsistemas.

En [Aren96] se presenta SIMS, un sistema para la integración de datos desde múltiples fuentes de datos. En lugar de realizar una integración de los esquemas de las fuentes,

SIMS adopta un enfoque alternativo, en el cual, primero se define un *modelo de dominio* de la aplicación, y se describe cada fuente usando este modelo. En SIMS no hay una traducción fija del lenguaje de consulta al lenguaje de consulta que utilizan las fuentes: las fuentes son seleccionadas dinámicamente e integradas cuando la consulta es enviada. Esto permite manejar fuentes de datos dinámicas y poder incorporar nuevas fuentes, así como la pérdida de algunas que se estuvieran utilizando. El modelo de dominio es formalizado usando un lenguaje basado en clases llamado LOOM. La evolución de SIMS es ARIADNE. El proyecto Ariadne [Ashi97] se centra en el desarrollo de de metodologías y herramientas para la construcción rápida de de *agentes inteligentes*, similares a los wrappers, para extraer, consultar e integrar datos procedentes de Internet. Utiliza el Wrapper Generator Toolkit. Como modelo de datos para representar los esquemas utiliza un lenguaje para representación del conocimiento llamado Loom KR.

En [Levy95] se presenta el sistema *Information Manifold* (IM), desarrollado en los laboratorios AT&T. Está basado en un modelo de dominio enriquecido que nos permite describir varias propiedades de las fuentes de datos, que van desde las características físicas hasta la información que pueden contener. Esto permite al usuario disponer de muchas posibilidades para formular sus consultas y extraer información de las diferentes fuentes de una manera uniforme. En IM podemos identificar dos componentes que son usados para describir cada fuente: una *world view* y una *descripción de las fuentes de datos*. Para ambos, estas descripciones esencialmente se forman utilizando esquemas relacionales. Las capacidades de una fuente se describen como consultas sobre un conjunto de relaciones y clases. Las consultas se formulan contra el componente world view, y las respuestas implican extraer los datos desde diferentes fuentes. IM utiliza un lenguaje de consulta llamado datalog.

En [Chaw94] encontramos la descripción de un proyecto desarrollado en la Universidad de Stanford llamado TSIMMIS (The Stanford-IBM Manager of Multiple Information System). Proporciona herramientas para el acceso integrado a múltiples y diversas fuentes de información y repositorios. Cada fuente de datos está equipada con un wrapper, que encapsula la fuente y convierte los datos subyacentes a un modelo de datos común. En TSIMMIS se utiliza un modelo de datos orientado a objetos simples, llamado *Object Exchange Model* (OEM). Por encima de los wrappers existe otro componente llamado *mediators*. Cada mediators obtiene datos de uno o más wrappers o de otros mediators, refina esta información integrándola y resolviendo conflictos entre los datos provenientes de diferentes fuentes, y proporciona la información resultante al usuario o a otros mediators.

En [Born99] para la representación e intercambio de información referente a las fuentes utilizan metadatos y para la interpretación de los metadatos proponen la utilización de *ontologías*, un vocabulario común para integrar datos procedentes de diferentes fuentes. En este caso las fuentes deben pertenecer a un dominio común. Se establece una relación entre los datos de una determinada fuente y lo que realmente representan en la vida real.

A cada dato o conjunto de datos de la fuente se le asigna un término de la ontología y se le añade información necesaria para describirlo en el modelo de datos empleado.

2.2.3.3 Integración de los datos materializados

El mantenimiento de vistas (o de los datos materializados) es un aspecto central en este contexto, y la forma de mantenimiento afecta al tiempo de respuesta y disponibilidad de los datos. En particular, como veremos más adelante, la actualización del almacén de datos es una tarea importante y costosa.

Aunque el tercer enfoque es el más apropiado para trabajar con almacenes de datos los dos primeros aportan algunos conceptos e ideas que son utilizados en el contexto del tercero.

Para la integración de los datos materializados comentaremos dos proyectos: el proyecto Squirrel y el proyecto WHIPS.

El proyecto Squirrel desarrollado en la Universidad de Colorado [Zhou96], es un sistema que proporciona un marco para la integración de datos, basada en el concepto de *integration mediator*. Estos componentes son módulos activos que soportan integración de datos usando una mezcla de los enfoques de datos virtuales y datos materializados.

Los mediators se definen como componentes software implementados sobre vistas materializadas e integradas sobre diferentes fuentes. Una característica de tales componentes es su habilidad para el mantenimiento de las vistas integradas, basándose en la capacidad activa de las fuentes de datos. En concreto, al inicializarse, el mediator envía a las fuentes de datos (bases de datos) una especificación de la información que necesita para actualizar incrementalmente el mantenimiento de las vistas, y espera que las fuentes, activamente, proporcionen dicha información. Este tipo de fuentes son llamadas *cooperativas*.

La arquitectura de Squirrel está compuesta de tres componentes: un conjunto de reglas activas, un modelo de ejecución para dichas reglas y un plan de descomposición de vistas (PDV). El PDV especifica las clases que el mediator mantiene, y proporciona la estructura básica para soportar el mantenimiento incremental.

Los integrators toman como entrada una especificación del mediator, expresada en un lenguaje de alto nivel llamado *Integration Specification Language* (ISL). Una especificación en ISL incluye una descripción de los subesquemas importantes de la fuente, y el criterio para emparejar dichos objetos con las correspondientes clases. La salida de este módulo es una implementación del mediator en el lenguaje *Heraclitus*, que es un lenguaje de programación de bases de datos, cuya principal característica es la habilidad de representar y manipular colecciones de actualizaciones sobre el estado actual de la base de datos.

El proyecto WHIPS (WareHouse Information Prototype at Stanford), es un sistema de almacén de datos desarrollado en la Universidad de Stanford [Hamm95], [Labi97]. La

arquitectura está formada por un conjunto independiente de módulos implementados como objetos CORBA. Esto asegura la modularidad y escalalibidad. El componente central del sistema es el *integrator*, que coordina al resto de módulos y al que el resto de módulos informan, y cuyo principal papel es facilitar el mantenimiento de las vistas. Cada fuente puede utilizar un modelo de datos diferente al utilizado en el almacén de datos. El modelo relacional se utiliza como modelo de unificación, y para cada fuente y el almacén de datos, los datos subyacentes se convierten al modelo relacional a través de un wrapper específico. Los cambios producidos en las fuentes son detectados por los monitores. Actualmente se centran en fuentes cooperativas, en concreto en bases de datos relacionales que activan trigger cuando hay alguna modificación y en ficheros que proporcionan instantáneas periódicas de las fuentes de datos. Para mantener la consistencia entre las fuentes y el data warehouse utiliza los algoritmos Strobe [Zhug96].

Las vistas se definen por el administrador del sistema como un subconjunto de SQL que incluye vistas select-project-join. La definición de vistas se pasa al módulo view-specifier, que la convierte en una estructura interna llamada *view-tree*, que incluye información proporcionada por el almacén de metadatos. El *view-tree* se envía al integrator, el cual lanza un *view-manager*, que es el responsable de manejar las vistas en el almacén de datos. Los cambios en cada fuente son detectados por un módulo llamado monitor que informa al integrator, que a su vez notifica los cambios de interés al *view-manager*. Las actualizaciones que considere necesarias el *view-manager*, se realizan pasando la consulta apropiada a un módulo global llamado query processor. Las respuestas devueltas por las fuentes, son ajustadas y combinadas de acuerdo a algoritmos de mantenimiento y consistencia de vistas [Zhug96] y enviadas a un módulo que se encarga de actualizar el almacén de datos (wrapper del almacén de datos).

2.3 El problema de la Integración en el Almacén de Datos

Cuando se diseñaron las aplicaciones existentes no sé hizo pensando en el futuro problema de la integración [Inmo02]. Cada aplicación tenía sus propios requerimientos de diseño y sus propios datos. No sorprende por tanto que existan los mismos datos en varias aplicaciones con diferentes nombres, datos etiquetados lo mismo en diferentes lugares, datos en la misma aplicación con el mismo nombre pero con diferentes medidas, etc. Por tanto, Extraer datos de diferentes fuentes e integrarlos de forma homogénea no es una tarea fácil de realizar.

Un ejemplo sencillo del problema de la integración son los datos que no están codificados de forma consistente, como podemos ver en la figura XXX respecto a la codificación de género. En una aplicación, género se codifica como m/f. En otra se codifica como 0/1. Y en otra como x/y. El problema no es cómo se codifica en cada una de esas aplicaciones, que de manera independiente puede resultar consistente, la cuestión es en el momento en que esos datos se pasan al almacén de datos. Es en ese momento cuando los diferentes valores deben ser integrados y almacenados con el valor adecuado en el almacén de datos.

Otro ejemplo podría ser el siguiente. Un campo en el que se almacenan medidas expresadas en diferentes formatos. En una aplicación las medidas se toman en pulgadas, en otra en centímetros, en otra en yardas, etc. Conforme los datos de cada una de esas aplicaciones se pasan al almacén de datos se van convirtiendo a un único formato.

La integración de los sistemas existentes no es la única dificultad en la transformación de datos desde los sistemas operacionales. Otro problema considerable es la eficiencia en el acceso a los datos de dichos sistemas. Por ejemplo, ¿cómo el programa que monitoriza los sistemas existentes sabe si un fichero ha sido monitorizado previamente?. Los sistemas actuales generan gran cantidad de datos e intentar escanear todos ellos cada vez que es necesario refrescar el almacén de datos es una tarea que consume mucho tiempo y recursos y que en algunas ocasiones puede resultar inútil.

Uno de los problemas más complejos de la integración y transformación es cuando disponemos de múltiples fuentes para un solo elemento. En otras palabras, en el almacén de datos hay un dato *a* que tiene como fuente un dato *b* de una aplicación BCD, un dato *c* de una aplicación CDE y un dato *d* de una aplicación DEF. Para complicar más el problema, la aplicación BCD está en un IMS, la aplicación CDE está en una base de datos relacional y la aplicación DEF está en Adabas.

En el caso de disponer de muchas fuentes para un solo dato, necesitamos saber la manera de integrar dichos datos. Como por ejemplo:

- El valor por defecto de a es 0.
- Cuando BCD.cantidad<0 entonces a = b.
- Cuando CDE.suma total<150 & mes=febrero, entonces a=c/12.
- Cuando año<1998 y DEF.stock<1500 entonces a=(d*DEF.cantidad).

Pueden existir muchas complicaciones bajo esas circunstancias. La primera es que ocurre si se satisface más de una condición. El diseñador debe decidir cual elegir o establecer un orden de prioridades.

La segunda complicación es acceder a todas y cada una de las fuentes de datos para poder determinar el valor correcto de *a*. En este caso, un problema sería disponer de todos los datos de las fuentes en el momento adecuado. Puede ser que los datos no estén accesibles online, o que estén disponibles los de una fuente pero no los de otra.

Una tercera complicación es el seguimiento y documentación de cómo se ha formado el dato *a*. Es posible que más adelante el analista necesite saber cual es el origen de *a*, a partir de qué dato se formó y cual fue la lógica empleada para derivar *a*.

Una cuarta complicación es la especificación de datos disponibles temporalmente como parte de las condiciones requeridas para calcular *a*. Por ejemplo, supongamos que un balance se crea semanalmente, se usa para determinadas operaciones y se borra. Supongamos también que el cálculo de *a* depende de algunos de los datos calculados

previamente en ese balance. Esto significa que el valor de *a* sólo puede ser calculado cuando el balance semanal está disponible.

Existen otros muchos problemas y variables que pueden ser necesarios para la integración y transformación de los datos provenientes de diferentes fuentes para ser incorporados al almacén de datos.

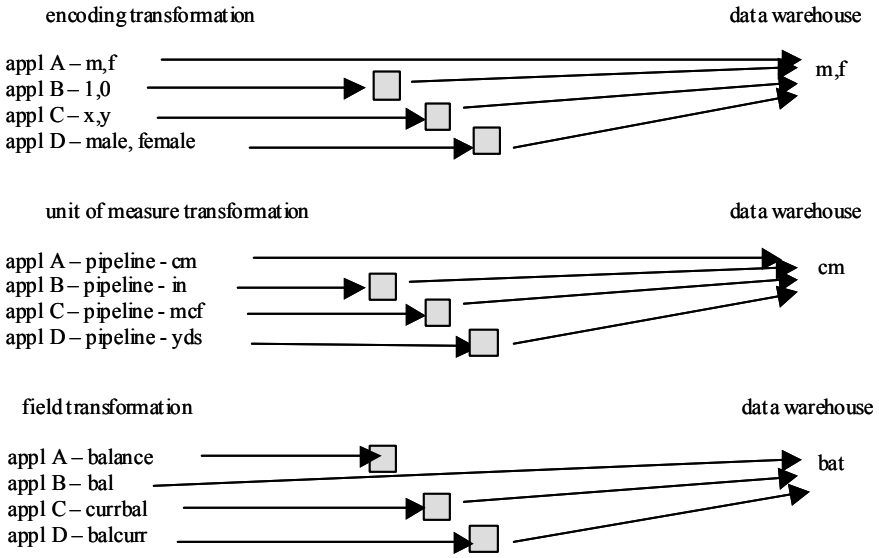

Figura 4. Problemas de integración

2.3.1 El papel del elemento tiempo

El factor tiempo es una variable esencial en los almacenes de datos. Podemos encontrar en "A Typical Data Warehouse Design Review" [Inmo02] que consta de 71 items, algunos relacionados con la cuestión temporal de los datos. Hemos seleccionado los que pueden implicar cuestiones temporales y deben ser tenidos en cuenta:

1. ¿Tenemos identificada la frecuencia del proceso de extracción (desde los sistemas operacionales hasta el almacén de datos)? ¿Cómo podremos identificar los cambios en el sistema operacional desde la última vez que se ejecutó el proceso de extracción?

- Mirando a los datos que tienen una marca de tiempo?
- Cambiando el código de la aplicación del sistema operacional?
- Consultando un fichero log? O un fichero de auditoria?
- Consultando un fichero delta?
- O comparando dos imágenes consecutivas del sistema operacional?

La frecuencia del proceso de extracción es un tema importante a tratar debido a los recursos que consume, la complejidad del mismo y los requerimientos temporales para llevarlos a cabo. Uno de los temas más complejos desde el punto de vista técnico, es determinar qué datos van a ser escaneados para su extracción. En algunos casos los datos que se necesitan pasar desde el entorno operacional al entorno del almacén de datos están claramente definidos, y en otros casos no está claro.

2. Qué volumen de datos están almacenados en el entorno DSS? Si el volumen es muy grande,
   - Se especificarán diferentes niveles de granularidad?
   - Los datos serán compactados?
   - Los datos se eliminarán periódicamente?
   - Los datos se moverán a almacenamiento secundario? Y con qué frecuencia?

3. Qué nivel de granularidad tendremos en el entorno del almacén de datos?
   - Un nivel alto?
   - Un nivel bajo?
   - Múltiples niveles?
   - Una de las cuestiones más importantes en el diseño del almacén de datos es la granularidad, y en concreto la posibilidad de disponer de múltiples niveles de granularidad.

4. Qué criterio de limpieza de datos tendremos? Serán los datos realmente eliminados o serán compactados y archivados? Hay algún tipo de requerimiento legal?

Aunque los datos guardados en el almacén de datos no sean necesarios en el momento actual, es recomendable compactarlos y almacenarlos en algún dispositivo por si fuera necesario su uso posterior. Dada la gran cantidad de datos y el crecimiento exponencial del volumen de éstos en la mayoría de los almacenes de datos, un aspecto importante del diseño es fijar criterios para la eliminación de datos del almacén y sumarización en dispositivos de almacenamiento secundarios.

5. Estarán permitidas actualizaciones en el almacén de datos? Porqué? Cuántas? Bajo qué circunstancias?

6. Qué intervalo de tiempo transcurrirá desde que se obtienen los datos en el sistema operacional hasta que están disponibles en el almacén de datos? Este intervalo será menor de 24 horas? Si es así, porqué y bajo qué condiciones? ?

7. Si el almacén de datos va a ser distribuido, tenemos identificadas todas las partes de dicho almacén? Cómo se van a manejar dichas partes?

8. Qué nivel de monitorización utilizaremos? A nivel de tabla? A nivel de fila? A nivel de columna? A nivel de transacción?

9. Es el nivel de granularidad del almacén de datos suficientemente bajo para dar servicio a todos los componentes adicionales que serán alimentados por dicho almacén?

10. Si el almacén de datos se va a utilizar para almacenar datos de negocio y datos de clickstream, con qué criterio filtrará los datos el manejador de granularidad?

El entorno Web genera gran cantidad de datos. En ocasiones estos datos están a un nivel de granularidad demasiado bajo y es necesario sumarizarlos y agregarlos antes de que sean pasados al almacén de datos. Esta tarea la realiza el manejador de granularidad.

Como podemos apreciar existen muchas preguntas relacionadas con el factor tiempo que es necesario tenerlas en cuenta a la hora de realizar la integración de los datos. Sin embargo, estas preguntas no abordan la problemática temporal relacionada con los métodos de extracción que será abordada a lo largo de la tesis.

2.4 Arquitectura de referencia

Cada vez con más frecuencia, con múltiples sistemas de bases de datos a los que convendría acceder de un modo integrado, es decir, como si se tratase de una sola base de datos. Si el acceso no es integrado, hace falta acceder separadamente a cada uno de los sistemas de bases de datos (para lo cual hay que conocer qué datos contiene, en qué forma, y uno de sus lenguajes de acceso), y luego combinar las respectivas respuestas (pasándolas a un formato común y eliminando redundancias).

Además, esta creciente necesidad del acceso integrado no sólo se presenta para datos almacenados en sistemas de bases de datos, sino también para otras fuentes de información: hojas de cálculo, páginas de la World Wide Web, servidores públicos de información, y otros sistemas de información "cerrados", es decir, capaces de responder a ciertas preguntas pero que "esconden" el formato de sus datos y no permiten su acceso directamente.

Este acceso integrado puede hacerse a) *en tiempo real*, es decir, que la consulta se procesa accediendo directamente a las bases de datos preexistentes, que ínter operan formando un sistema cooperativo, o bien b) consultando un *almacén de datos*, en el que periódicamente se vuelcan y consolidan los datos de esas bases de datos.

2.4.1 Arquitectura de Sheth y Larson

Un sistema de bases de datos federado (FDBS) está formado por diferentes sistemas de bases de datos componentes; proporciona acceso integrado a dichas bases de datos componentes: ellas cooperan (inter-operan) una con otra para producir una respuesta consolidada a las consultas definidas sobre el FDBS. Generalmente, los FDBS no contienen datos almacenados, las consultas se responden accediendo a los sistemas de las bases de datos componentes. Se han propuesta muchas arquitecturas de FDBS para

proporcionar acceso integrado a las fuentes, nosotros partimos de la propuesta por Sheth y Larson [Shet90].

En la Figura 5 se distinguen tres módulos (rodeados por cajas), cada uno se corresponde con un modelo de datos: bases de datos componentes (modelos nativos), FDBS (modelo de datos canónico) y esquemas externos (modelos de usuario).

Sheth & Larson proponen una arquitectura de cinco niveles. Es muy general y abarca la mayoría de las arquitecturas existentes (Figura 5). Existen tres tipos de modelos de datos: primero, cada base de datos componente tiene su propio modelo de datos nativo; segundo, un modelo de datos canónico (MDC) que lo utiliza el FDBS; y tercero, cada esquema externo se puede definir en diferentes modelos de usuario. Además, se definen tres mecanismos que se utilizan para definir esquemas en los diferentes niveles a partir de los de niveles inferiores. Los mecanismos son: *modelos de traslación* (entre el MCD y otros modelos), *integración de esquemas* (todos ellos definidos usando el MCD) y *definición de esquemas derivados* a partir de un esquema.

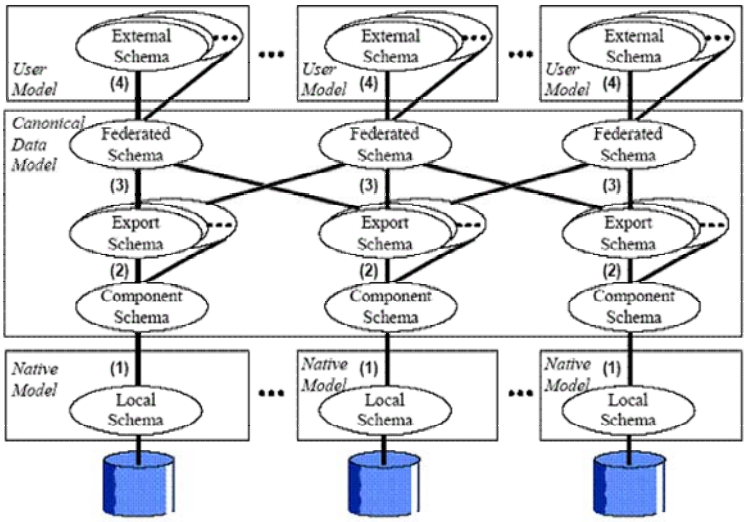

Figura 5. Arquitectura de 5 niveles de esquemas de Sheth & Larson

Como se aprecia en la Figura 5, cada base de datos componente tiene su propio esquema local definido en su modelo nativo. Los esquemas locales se traducen (marca 1) para obtener esquemas componentes definidos usando el MCD. Cada base de datos negocia que parte de sus datos (o esquema) es accesible, por tanto, utilizando el mecanismo de definición de esquemas derivados se definen diferentes esquemas de exportación a partir de cada esquema componente (marca 2). Cada esquema de exportación representa el resultado de una negociación. Los esquemas de exportación se integran (marca 3) para formar el esquema federado. A partir de cada esquema federado se definen los esquemas externos en diferentes modelos usando los mecanismos de definición de esquemas derivados y los de traslación de modelos (marca 4).

2.4.2 Arquitectura integrada de BD federadas y almacenes de datos.

Aunque se reconocen importantes elementos en común entre los almacenes de datos y otras áreas previamente estudiadas, no se aprovechan en su totalidad los avances conseguidos en estas áreas para su aplicación a la construcción de almacenes de datos y data marts, en concreto, nos referimos especialmente a las BD federadas, BD temporales, así como a la definición de esquemas externos.

En trabajos anteriores se amplió la arquitectura de S&L para proponer una arquitectura integrada de BD que incorpora el concepto de almacén de datos y de data mart [Samo98] [Samo99]. En esta arquitectura, el almacén de datos y los data marts forman parte de la BD, son una funcionalidad más de la BD en lugar de ser sistemas independientes construidos utilizando una BD. A diferencia de otras propuestas, la nuestra permite que conceptos de BD federadas, temporales y orientadas a objetos, así como de definición de esquemas externos, puedan ser aplicados directamente a la definición y construcción de almacenes de datos. Según la arquitectura de BD definida, el esquema de los almacenes de datos se define usando el MCD; sin embargo, los usuarios pueden trabajar sobre los almacenes de datos usando otros modelos de datos, según sus necesidades.

En la Figura 6 se muestra nuestra arquitectura integrada considerando un solo esquema federado para simplificar la representación. El *esquema del almacén de datos* y los *esquemas de data marts* se sitúan en paralelo con el resto de esquemas y pueden ser definidos a partir de cualquier esquema federado.

El esquema federado juega un papel similar al del ADO en la arquitectura propuesta en [Inmo98], con la gran diferencia de que sus datos no están materializados. En caso de existir aplicaciones que requieran un acceso más óptimo a los datos, éstos pueden materializarse, definiendo así el ADO a partir del esquema federado. El almacén de datos puede ser definido directamente tanto a partir del esquema federado como del ADO y, a diferencia del esquema federado, el almacén y los data marts se encuentran siempre materializados (ya que, por definición, almacenan la historia de los datos formada a partir de imágenes obtenidas de los sistemas operacionales).

En caso de necesitarse, la definición del *Esquema del Almacén de Datos Operacionales* se realizaría básicamente de la misma forma que se define el Esquema Federado, añadiendo al *Procesador de Integración* los elementos necesarios para decidir sobre el almacenamiento de los datos (en la Figura 6 se ha representado ambos esquemas pero tan solo se necesita uno de ellos para definir el esquema del almacén de datos).

El Esquema del almacén de datos se define mediante el *Procesador de Almacén de Datos* que contempla las características de estructuración y almacenamiento de los datos del almacén, así como también ofrece la posibilidad de considerar los distintos niveles de seguridad definidos. Los diferentes Esquemas de data mart son derivados a partir del Esquema del almacén de datos mediante un *Procesador de Derivación* que, al igual que el

Procesador de Almacén de Datos, permite definir las características de estructuración y almacenamiento propias de los almacenes de datos.

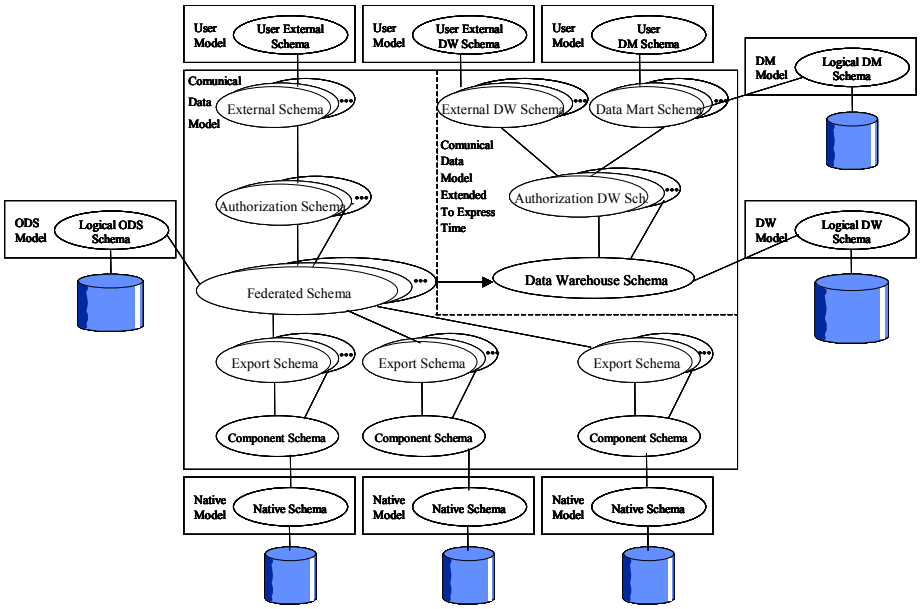

Figura 6. Arquitectura integrada de BD federadas y almacenes de datos.

Los esquemas del almacén de datos y de Data Marts son definidos usando el MCD; al igual que los Esquemas Traducidos, éstos pueden ser traducidos al modelo nativo de un usuario o grupo de usuarios federados mediante el Procesador de Transformación, obteniéndose así los Esquemas de almacén de datos de Usuario. En Figura 7 podemos ver la relación entre los diferentes esquemas así como los procesadores que se encargan de las conversión de los mismos.

La arquitectura propuesta integra tanto el acceso en tiempo real como los almacenes de datos, el resultado es un sistema cooperativo para la integración de fuentes heterogéneas de información y almacenes de datos.

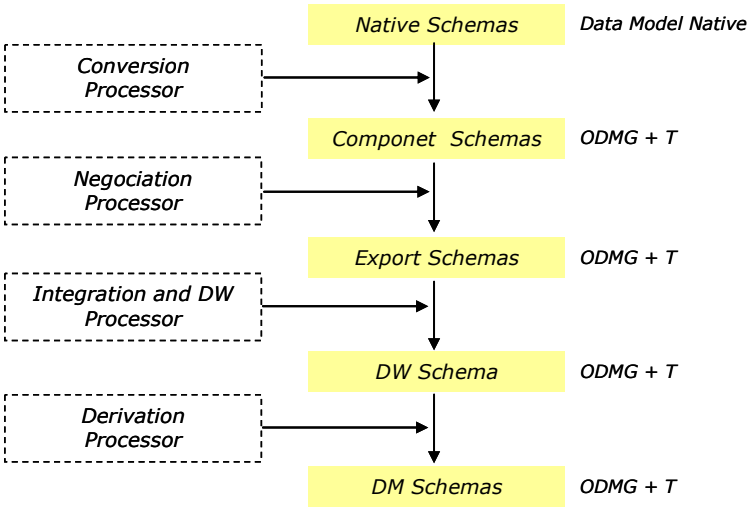

Figura 7. Conversión de esquemas

2.4.3 Trabajo del grupo de investigación

En este apartado se revisan algunos de los elementos más relevantes en los ha trabajado el grupo para llevar a cabo la implementación de la arquitectura integrada propuesta en los puntos anteriores.

2.4.3.1 Control de acceso cooperativo

Entre los distintos aspectos a considerar al realizar la integración se hace necesario abordar la protección de datos. En cualquier sistema de gestión de bases de datos esta protección puede obtenerse mediante las siguientes medidas de seguridad: control de flujo, control de inferencia y control de acceso. Evidentemente, para poder aplicar estas medidas de seguridad será necesario que el sistema disponga de los mecanismos necesarios para la identificación y autentificación de sus usuarios.

Nuestro sistema cooperativo basa su mecanismo de protección de datos, y más concretamente de control de acceso, en la técnica de sistema de seguridad multinivel ya que por su propio funcionamiento no solo ayuda al control de acceso sino también al control del flujo de la información. Debe integrar fuentes de información heterogéneas por lo que el acceso a la información solo puede permitirse dependiendo de los criterios de seguridad locales de dichas fuentes (ya que debe mantenerse su autonomía), pero también dependiendo de los criterios de seguridad globales del sistema cooperativo.

Pretendemos estudiar la forma de resolver las heterogeneidades para el caso de una fuente de información con sistema de seguridad basado en un control de acceso discrecional y el caso de una fuente de información con un sistema de seguridad según un control de acceso basado en los "roles".

Desde el punto de vista de la seguridad también tienen importancia los cambios que se puedan producir debidos a la evolución de las fuentes de información, así pues será necesario analizar los distintos cambios que puedan darse así como la forma de incorporarlos en el sistema de seguridad del sistema cooperativo en funcionamiento.

2.4.3.2 Gestión de transacciones cooperativas

Una de las funciones básicas de un sistema gestor de base de datos convencional es la de permitir que múltiples usuarios puedan interactuar a la vez con el sistema para obtener datos almacenados en la base de datos, bien ejecutando programas de aplicación, bien ejecutando consultas interactivas. Estas ejecuciones se traducen en un conjunto de operaciones sobre unidades de datos que recibe el nombre de transacción. A su vez, las transacciones son la unidad de trabajo fundamental de un sistema gestor de base de datos. Esta función se puede trasladar a sistemas cooperativos de fuentes heterogéneas de información, teniendo en cuenta que además de los usuarios locales a cada fuente, aparece la figura del usuario global del sistema cooperativo que deseará obtener en tiempo real datos almacenados en las distintas fuentes que integran el sistema cooperativo.

Por lo tanto, los distintos sistemas de bases de datos continuarán funcionando autónomamente unos de otros, prestando sus servicios a los programas y usuarios preexistentes al mismo tiempo que soportan la ejecución de las subconsultas derivadas de los accesos integrados al sistema cooperativo. Ello supone nuevos problemas de control de concurrencia y recuperación, una nueva definición de "transacciones" del sistema cooperativo (diferentes de las transacciones convencionales de los SGBDs), y un gestor de transacciones a nivel cooperativo capaz de gestionarlas adecuadamente.

2.4.3.3 Definición de esquemas externos y clases derivadas en BDOO

En nuestra propuesta, la superación de la heterogeneidad entre modelos de datos se consigue mediante un MCD, el modelo BLOOM, un modelo orientado a objetos. De ahí nuestro interés en investigar sobre clases derivadas y esquemas externos en Bases de Datos Orientadas a Objetos (BDOO), ya que estos son los mecanismos en los que se basa el Procesador de Derivación definido en nuestra arquitectura.

La arquitectura de tres niveles de esquemas: interno, conceptual y externo, de ANSI/SPARC ha sido aplicada extensamente en las BD relacionales. En BDOO, los esquemas interno y conceptual han sido estudiados profundamente; no así los esquemas externos, a pesar de que la independencia lógica de datos que ofrecen es también un requerimiento para las BDOO. Además, tan solo unos pocos trabajos hacen referencia la arquitectura ANSI/SPARC [ANSI86] y usan su terminología. En nuestros trabajos anteriores [Samo95] [Samo98a] ha sido definida de forma preliminar una nueva metodología de definición de esquemas externos tomando como base la arquitectura ANSI/SPARC.

2.4.3.4 Evolución del esquema en BDOO

Un sistema cooperativo no se construye de una vez por todas y luego opera sin más: no puede ser estático, sino que debe adaptarse a la dinámica de los componentes que integra, como se recoge en el reciente Cooperative Information Systems: A Manifesto [Mich97]. Los cambios en los esquemas son particularmente importantes, por lo que uno de nuestros objetivos es investigar sobre evolución de esquemas.

Los esquemas externos son derivados a partir del esquema conceptual de la BD, permiten presentar la información contenida en la BD de forma que se adapte mejor a las necesidades específicas de los usuarios finales en cuanto a su estructura y contenido. De cara a los usuarios finales, los esquemas externos pueden ser utilizados para simular algunos tipos de cambios en el esquema conceptual. La información en los esquemas externos ha de ser derivada a partir de la información en el esquema conceptual, por tanto, los cambios que los esquemas externos permiten simular son exclusivamente aquellos en los que toda la información del esquema resultante siempre se puede obtener a partir del esquema original: El esquema externo tiene la misma capacidad de información (las operaciones que lo producen son "capacity preserving transformations") o menor capacidad de información ("capacity reducing transformations") que el esquema conceptual [Tres93].

En ocasiones, la necesidad de información de los usuarios finales aumenta y pasan a necesitar nueva información que no puede ser derivada a partir de la información existente en la BD (esquemas que son resultado de "capacity augmenting transformations"). En BDOO, esta situación se soluciona modificando el esquema conceptual mediante la definición de nuevas clases o bien la modificación de clases previamente existentes para que incorporen dicha información. En este caso no se dispone de un medio que permita simular los cambios, sino que éstos afectan directamente al esquema conceptual. Por tanto, se dispone de menor flexibilidad que en el caso anterior, ya que realizar cambios en el esquema conceptual es más costoso que realizarlos exclusivamente en algún esquema externo. Ésta es una limitación de los sistemas actuales sobre la que trabajamos para subsanarla.

2.4.4 Ampliación de la arquitectura para incorporar elementos temporales

Partiendo de la arquitectura de S&L y la propuesta en [Samo98] proponemos la siguiente arquitectura de referencia para nuestro trabajo. En nuestra arquitectura, además de los cinco niveles de esquema propuestos en el marco de referencia de Sheth y Larson (1990), aparecen otros niveles. Los diferentes esquemas son:

1. *Esquema Nativo*: es el esquema conceptual de una base de datos componente expresado en su modelo de datos nativo.

2. *Esquema Componente*: es la conversión de un Esquema Nativo en nuestro MCD.
3. *Esquema Componente T*: es la conversión de un Esquema Nativo en nuestro MCD enriquecido para expresar conceptos temporales.
4. *Esquema de Exportación*: representa la parte de un Esquema Componente que está disponible a un conjunto de usuarios federados.
5. *Esquema de Exportación T*: representa la parte de un Esquema Componente T que está disponible al diseñador del almacén. Está expresado en el mismo MCD que el Esquema Componente T.
6. *Esquema Federado*: es la integración de múltiples Esquemas de Exportación, cada Esquema Federado soporta una semántica.
7. *Esquema externos*: de usuarios de la BDF, del DW o del DM.
8. *Esquema del almacén*: es la integración de múltiples Esquemas de Exportación T según las necesidades del diseño del mismo expresado en un MCD enriquecido para expresar conceptos temporales.
9. *Esquema de Usuario*: es la conversión de un Esquema Traducido en el modelo de datos nativo del usuario.

Los esquemas que son de interés para esta tesis son: el Esquema componente T, el Esquema de Exportación T y el Esquema del Almacén construido a partir de los Esquemas de Exportación T.

Primero debemos enriquecer un modelo de datos OO con elementos temporales. La representación diferenciada del tiempo, dentro del esquema del Almacén, facilitará la comprensión y manejo de sus contenidos. Una vez que se disponga de la representación adecuada para los elementos temporales del almacén, habrá que establecer los mecanismos de definición de su esquema de datos. La información que puede contener siempre está condicionada por los datos disponibles en las fuentes de datos (base de datos componentes). Se trata de determinar qué información se puede extraer de esas bases de datos componentes y cómo queda representada en el almacén de datos, según los requerimientos del problema a resolver.

Por otra parte, La extracción de datos de las bases de datos componentes de una federación se hace bajo demanda. Es decir, cada consulta de los usuarios federados se traduce en las consultas correspondientes sobre las bases de datos componentes. Con el almacén de datos esto ya no se pueda realizar así. En este caso, la extracción de información hay que hacerla por adelantado, sin esperar las consultas de los usuarios. De esta manera, se hace necesario el estudio de la representación de los elementos (basados en conceptos de BD de Tiempo Real y BD Temporales) que permitan expresar las características de obtención de los datos de las distintas fuentes de información. En nuestro caso se generaría un *Esquema Componente T* (ECT) enriquecido con las

características temporales de las fuentes. Podemos decir que se encargaría de "exportar" las características temporales de las fuentes de datos.

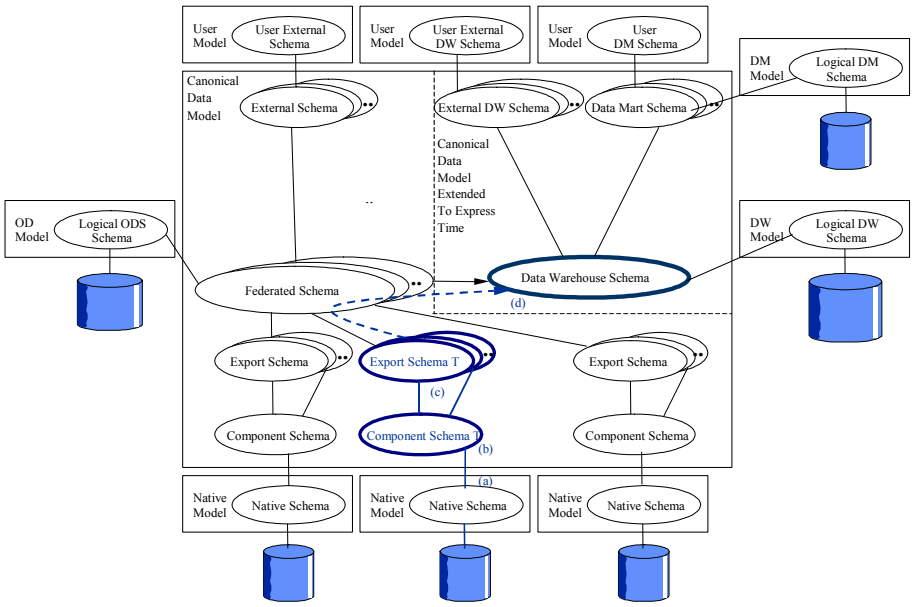

Figura 8. Ampliación de la arquitectura de Sheth y Larson con esquemas temporales.

A partir del ECT también podría generarse un *Esquema de Exportación T* (EET) con la parte del esquema que se quiere hacer visible para ser utilizado posteriormente en la definición del esquema del almacén de datos. Por tanto, necesitaremos mecanismos de almacenamiento de la información necesarios para la extracción de información por adelantado de las bases de datos componentes.

2.5 Construcción del Sistema Federado

A continuación vamos a comentar la arquitectura de la que se parte y que ha sido desarrollado previamente en otros trabajos [Cast93] [Cast94] [Garc95]. Se trata de una arquitectura diseñada para dar soporte a un sistema federado, pero que aquí se adapta para aplicarla a la construcción de un almacén de datos. La diferencia principal entre una y otra es que en este caso los datos integrados se encuentran materializados.

2.5.1 Introducción

En la Figura 9 se detallan los diferentes componentes de la arquitectura. Pueden distinguirse dos etapas diferentes. Una etapa se encarga de resolver las heterogeneidades sintácticas, mientras que la otra es la encargada de resolver las heterogeneidades semánticas.

La primera etapa de la metodología se denomina *Enriquecimiento semántico*. En esta etapa queda superado el problema de la heterogeneidad sintáctica gracias a la utilización de un modelo canónico de datos, al que son convertidas todas las Bases de Datos Componentes (BDC). La segunda etapa, se denomina *Integración de esquemas de datos*. El problema de la heterogeneidad semántica se resuelve mediante la *detección de similitudes* entre objetos y su posterior integración.

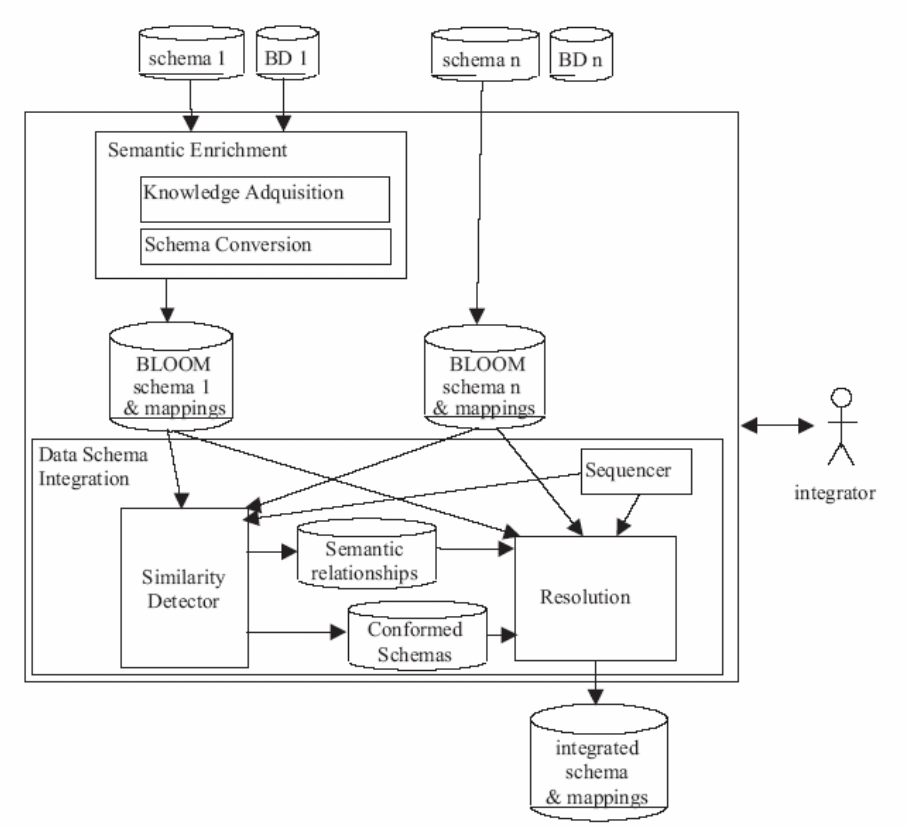

Figura 9. Componentes de la arquitectura

Además de la descripción de las metodologías correspondientes a cada una de las dos etapas, en la sección siguiente se describe con mayor detalle la arquitectura de construcción en la que se basan estas metodologías. En el capítulo 3 de la tesis se describe en mayor profundidad el modelo canónico de datos utilizado.

2.5.2 Arquitectura de construcción

La construcción del sistema federado se debe realizar a partir de una arquitectura adecuada que soporte la interoperabilidad entre las BDCs. Las metodologías empleadas para llegar a construir el sistema federado que son descritas se basan el la arquitectura de referencia de Sheth y Larson. Se trata de una arquitectura general que engloba la mayoría

del resto de arquitecturas propuestas para este mismo objetivo que se pueden encontrar en la literatura. Además, cuando se construye un sistema federado fuertemente acoplado cabe adoptar un modelo canónico de datos para la federación. En nuestro caso se utilizará ODMG enriquecido con elementos temporales.

2.5.3 ODMG como MCD

Originalmente se utilizaba BLOOM (BarceLona Object-Oriented Model) como MCD de la federación, una extensión semántica de un modelo orientado a objetos. En este trabajo ha sido sustituido por ODMG como modelo de datos canónico.

2.5.4 Enriquecimiento semántico

El *enriquecimiento semántico* es el proceso que permite integrar bases de datos, que utilizan un modelo de datos tradicional, en el sistema que se pretende construir. El enriquecimiento semántico realiza una translación del esquema de datos de las BDC al modelo canónico de la federación, que como ya se ha visto es ODMG.

La metodología de enriquecimiento semántico está descrita en [Cast94], [Cast93]. En concreto consta de dos fases: la fase de *adquisición de conocimiento* y la *fase de conversión*. En la fase de adquisición de conocimiento se descubren las relaciones semánticas que están implícitamente expresadas en forma de diferentes tipos de restricciones en el esquema de las BDC. En la fase de conversión el esquema de las BDCs es aumentado con la información semántica obtenida en la primera fase y transformado en una representación más rica semánticamente expresada en el MCD, donde la información semántica implícita se hace explícita. A continuación se describe con mayor profundidad cada una de estas fases.

2.5.4.1 Fase de adquisición del conocimiento

El procedimiento de enriquecimiento semántico pretende obtener descripciones más ricas de los objetos a integrar. La semántica de estos objetos viene dada principalmente por las relaciones que tienen entre otros objetos. Esta primera fase se centra en descubrir la naturaleza de estas relaciones. Para lograr este objetivo es necesario realizar una serie de pasos:

Paso 1: *Inferencia de claves*

En este paso se obtienen las claves, tanto las simples como las compuestas, de cada relación. Pueden ser inferidas de diferentes formas teniendo en cuenta la información disponible. Si está especificada de alguna forma en la definición del esquema puede obtenerse consultando el diccionario de datos.

Paso 2: *Inferencia de dependencias funcionales*

A partir de las dependencias funcionales de las BDCs también es posible extraer información semántica. Este tipo de dependencias permite realizar la normalización del esquema resultante. Se utiliza el algoritmo LHS → RHS (Left Hand Side determina funcionalmente Right Hand Side) para descubrir estas funcionalidades. Este algoritmo infiere un conjunto de RHSs que posteriormente hay que comprobar para asegurarse que verifican cada restricción funcional.

Paso 3: *Normalización*

En muchos casos, las bases de datos relacionales de las fuentes de datos no están correctamente normalizadas, por no haber sabido distinguir correctamente los diferentes conceptos en el momento de su diseño. En este paso se procede a su normalización en la tercera forma normal, lo que ayuda en la detección de relaciones entre diferentes BDs.

Paso 4: *Determinación del tipo de los identificadores*

En este punto se analizan todas las cláusulas para determinar la corrección a la hora de decidir su clave primaria. En algunas situaciones puede que la clave elegida identifique la relación correctamente, pero no satisfaga la propiedad de unicidad. Desde el punto de vista semántico no representan entonces identificadores adecuados.

Paso 5: *Inferencia de las dependencias de inclusión*

Consiste en obtener todas las dependencias de inclusión (DINs) que componen las BDs de las fuentes. Una DIN indica que RHS incluye LHS. El método más sencillo para comprobar todas las posibles DINs es comprobarlas una a una, pero no es posible aplicarlo en la práctica. Se deben utilizar heurísticas que reduzcan el conjunto de DINs a comprobar.

Paso 6: *Inferencia de las dependencias de exclusión y complementariedad*

Este tipo de dependencias se utilizan en la segunda fase de la metodología de enriquecimiento semántico para determinar el tipo de cada especialización (alternativa, complementaria, disjunta y general).

2.5.4.2 Fase de conversión de esquemas

En esta segunda fase del enriquecimiento semántico se convierten los esquemas de las fuentes de datos en esquemas más ricos, semánticamente hablando, expresados en el MCD. Para ello se utilizan los resultados obtenidos en la fase de adquisición de conocimiento. Consta de los siguientes pasos.

Paso 1: *Análisis de las DINs*

Las relaciones estructurales semánticas forman un entramado de conexiones entre las clases y es posible definir de forma precisa la naturaleza de estas conexiones en la descripción de una clase para expresar toda la información semántica que incluye. Se puede dividir a su vez en dos tareas diferentes: detección de las entidades abstractas e

identificación de las abstracciones semánticas. La primera de ellas permite incluir entidades que existen pero que no han sido explícitamente recogidas en los esquemas de las fuentes de datos. En la segunda subtarea se procede a identificar con mayor precisión las relaciones semánticas que forman el entramado de conexiones entre los objetos de las BDs.

Paso 2: *Procesamiento de los atributos restantes*

En este paso se tratan todos los atributos que no aparecen en la RHS de las DINs y todos los atributos que no están involucrados en DIN. Se añaden a las clases correspondientes conectándolos mediante la abstracción de Agregación.

Paso 3: *Creación de las clases*

Este paso, más que ser un paso en si mismo, refleja el resultado final de la fase de conversión de esquemas, cuando se obtienen todas las clases con sus relaciones y forman el esquema en el MCD correspondiente al esquema de la fuente de datos que se ha enriquecido.

2.5.5 Integración de esquemas de datos

La integración de los esquemas de datos, de forma coherente, consistente y completa es una tarea difícil. Implica un conocimiento profundo de la semántica de las BDCs para identificar los objetos que representan conceptos similares e integrarlos de acuerdo a estas relaciones semánticas. Si ya de por sí esta tarea es lo suficientemente compleja, se complica aun más al tener que resolver conflictos semánticos entre diferentes fuentes de datos. En esta sección se describen brevemente las fases de *Detección de similitudes* y *Resolución* de la metodología de integración de esquemas de datos.

Los esquemas enriquecidos, o los esquemas de las BDCs en el MCD, son la entrada al módulo Detector de similitudes, encargado de detectar las similitudes entre los objetos y sus relaciones semánticas. El resultado obtenido debe ser confirmado por el administrador del sistema. Una vez que se han identificado las relaciones semánticas, el módulo de Resolución se encarga de resolver los conflictos semánticos entre los objetos. El *secuenciador* se encarga de controlar la secuencia de pares de esquemas que deben integrarse. A continuación se detallan con mayor profundidad los dos módulos introducidos.

2.5.5.1 Detección de similitudes

El objetivo de esta fase es identificar los objetos relacionados semánticamente. La detección de similitudes se realiza mediante un proceso de comparación. El problema principal no es la comparación en sí misma, sino la cuantificación de las similitudes.

Este proceso de detección se divide a su vez en dos partes: una sobre la estrategia de detección y otra sobre el criterio de comparación de clases.

2.5.5.1.1 La estrategia a nivel grueso

Está guiada por las estructuras definidas a lo largo de la dimensión de generalización, que determina la secuencia de comparaciones de especialización. Siempre que una clase de una BDC resulte similar a otra clase de otra BDC se invoca al *Analizador de generalización* para comparar las especializaciones de ambas. Tiene en cuenta los diferentes tipos de abstracciones de la especialización. La especialización de una clase C1 con un grupo de subclases se compara con la especialización de una clase C2 si coinciden en el tipo, el grado (número de subclases) y el efecto que producen al ser eliminadas. Teniendo en cuenta estos factores se le aplica un valor de penalización al grado de similitud entre las especializaciones y sus subclases.

2.5.5.1.2 La estrategia a nivel fino

Una vez que el *Analizador de generalizaciones* ha identificado los grupos de clases (especializaciones) a comparar, el *Analizador de agregaciones* identifica los pares de clases estos dos grupos que han de compararse. En este sentido se utiliza la potencia de las abstracciones de la dimensión de agregación. En primer lugar se analizan las composiciones y después las agregaciones simples. En este nivel de la estrategia no se tiene en cuanta la abstracción mas fuerte de cada clase. La idea es que si dos clases tienen la misma abstracción y además la abstracción es fuerte hay más posibilidades de que las clases sen similares. Si las dos clases comparadas no resultan ser similares se continúa buscando una clase que pueda ser similar a la más fuerte. Por cada abstracción de agregación si hay un conjunto de relaciones posibles, cada una produce una penalización particular en el caso de las relaciones aplicadas a la abstracción de especialización.

2.5.5.1.3 El criterio

El *criterio de similitud* también se basa en la dimensión de agregación. El criterio indica cuales de los aspectos de las clases se han de comparar y el grado de similitud entre las clases. Cada abstracción tiene asignado un peso particular teniendo en cuenta lo fuerte que es semánticamente. La función que calcula el grado de similitud se basa en el peso de las abstracciones, la penalización y la similitud entre las clases que componen las que se están comparando.

2.5.5.2 Resolución

En esta fase los esquemas de exportación se integran de acuerdo con sus relaciones semánticas detectadas en la fase de detección. De hecho, la resolución de los conflictos semánticos comienza a realizarse en la fase de detección, en el momento en el que se comienzan a aplicar relaciones para obtener las clases que son comparables (*conformación*). Las correspondencias entre esquemas, derivados a partir de esta conformación, se

almacenan temporalmente para que posteriormente sean utilizadas por el módulo de resolución.

El principal operador de integración utilizado es la *Generalización discriminada*. Se trata de una generalización en la que cada instancia se etiqueta con un discriminante que coincide con el nombre de su BDC fuente. Las ventajas que se obtienen con esta operación son:

- La opcionalidad de la transparencia de BD.
- No se produce pérdida de información.
- Se da soporte a múltiples semánticas.

2.6 Arquitectura detallada

Pasamos a describir a continuación la arquitectura funcional presentada en la sección 2.4.4 más detalladamente. En la Figura 10 podemos ver las etapas de la metodología:

Esquemas Nativos

Inicialmente tenemos las diferentes fuentes de datos expresadas en su esquema nativo. Cada fuente tendrá, un esquema, los datos propios de la fuente y los metadatos. En los metadatos tendremos información relevante sobre la fuente: datos sobre el esquema, metadatos temporales sobre disponibilidad de la fuente, disponibilidad del log, delta si los tuviera, etc.

Preintegración

En la fase de Preintegración se realiza el enriquecimiento semántico de los esquemas nativos de las fuentes por parte del procesador de conversión. En esta fase se utilizan los metadatos temporales de la fuente de datos para enriquecer al esquema de la fuente con elemento temporales. Como resultado se obtiene el esquema componente (ECT) expresado en el MCD, en nuestro caso ODMG enriquecido con elementos temporales (ODMGT). Esta fase se explica con más detalle en el Capítulo 2.

Esquemas Componentes y de Exportación

A partir de los esquemas componentes expresados en ODMGT el *procesador de negociación* genera los esquemas de exportación (EET) expresados en ODMGT. Estos EET son la parte de los ECT que se considera necesaria para su integración en el almacén de datos. Por motivos de seguridad, de privacidad o simplemente porque no sean necesarios para el problema a resolver, parte del ECT se puede ocultar. Los EET son esquemas externos de los ECT. En los Capítulos 3, 4 y 5 se explican con más detalle este proceso.

Integración

A partir de los EET de diferentes BDs componentes se construyen el esquema del almacén de datos (expresado el ODMGT). Este proceso se realiza por un *Procesador de Integración* que sugiere cómo integrar los *Esquemas de Exportación* ayudando a resolver heterogeneidades semánticas. En la definición del *Esquema del Almacén de Datos* también participa el *Procesador de Almacén de Datos* que contempla las características de estructuración y almacenamiento de los datos del almacén.

A la arquitectura de referencia se le han añadido dos módulos necesarios para llevar a cabo la integración de las propiedades temporales de los datos teniendo en cuenta el método de extracción de datos utilizado, son el *Procesador de Integración Temporal* y el *Generador de Metadatos de Refresco*.

El procesador *Procesador de Integración Temporal* utiliza el conjunto de relaciones semánticas y los esquemas conformados obtenidos durante la fase de detección de similitudes de la metodología de integración de esquemas de datos. Como resultado se obtiene información en forma de reglas acerca de las posibilidades de integración que existen entre los datos provenientes de las fuentes (granularidad mínima, el periodo de refresco debe estar acotado entre algunos valores concretos,…). Esta información se guarda en el almacén de *Metadatos Temporales*. Como resultado del proceso de Integración Temporal se obtienen además una serie de *funciones de traslación*, que identifican los atributos de los esquemas de las fuentes de datos que se integran entre sí para obtener un atributo del esquema del almacén de datos.

El *Generador de Metadatos de Refresco* determina los parámetros más adecuados para llevar a cabo el refresco de los datos del esquema del almacén, generado a partir de la fase de resolución de la metodología de integración de esquemas de datos. Es en esta segunda fase donde, a partir de los requisitos mínimos generados por la primera fase de integración temporal y que se recogen en el almacén de metadatos temporales, el diseñador del almacén fija los parámetros de refresco de los datos almacenados en el almacén. Como resultado se obtiene el esquema del almacén de datos junto con los *Metadatos de Refresco* necesarios para actualizar el almacén de forma coherente en función del tipo de método de extracción y otras características temporales de las fuentes de datos a las que se accede.

La obtención del Esquema del Almacén de Datos y de los Esquemas de Exportación no es un proceso lineal; se necesita que los Procesadores de Integración y Negociación colaboren en un proceso iterativo donde participen los administradores locales y del almacén de datos [Oliv96]. El Capítulo 6 está dedicado a explicar con más detalle el proceso de integración temporal.

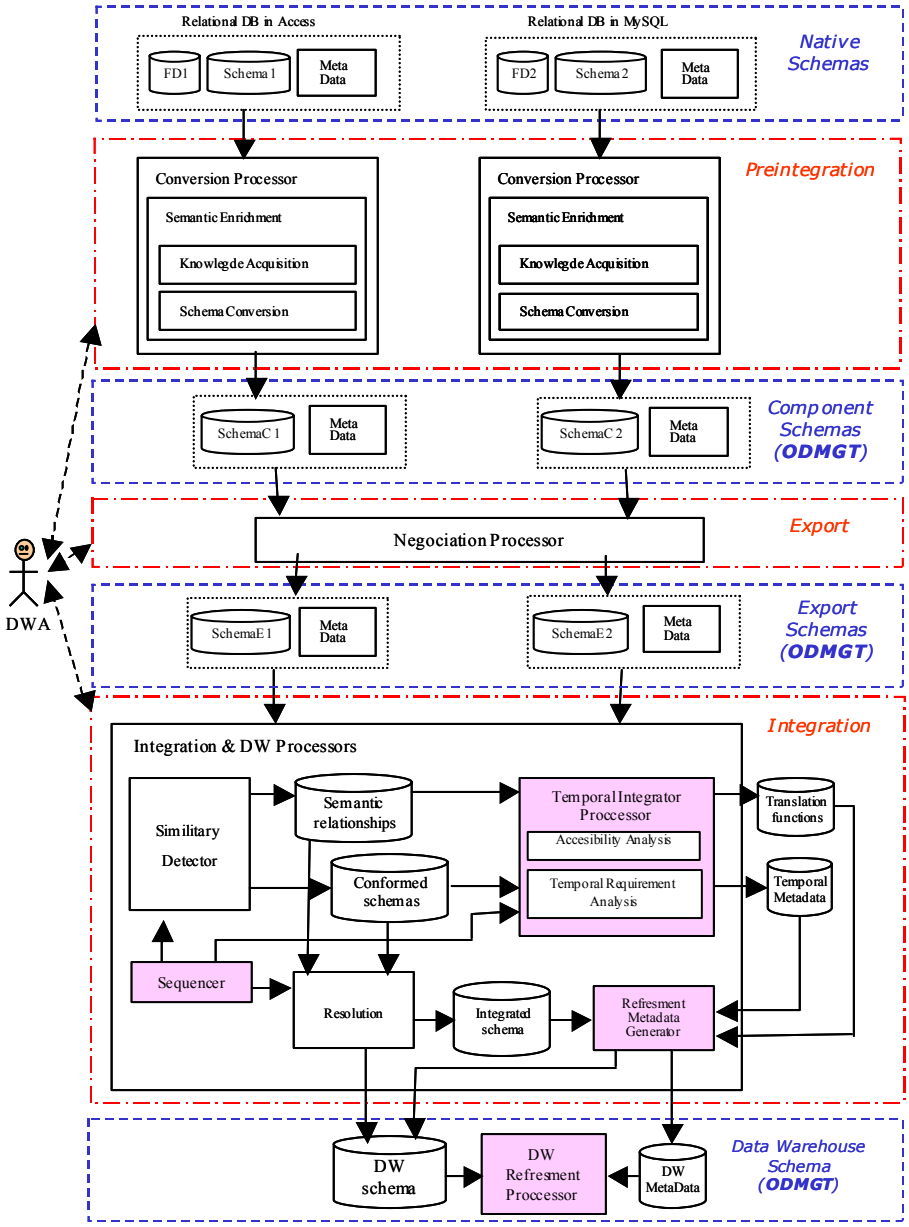

Figura 10. Metodología de integración

Esquema y Refresco del Almacén de Datos

A continuación de la integración temporal y una vez obtenido el esquema del almacén, es necesario el mantenimiento y actualización del mismo. Esta función la realiza el *Procesador de Refresco del Almacén*. A partir de los requisitos mínimos que se deben cumplir para llevar a cabo la integración entre dos datos de diferentes fuentes de datos, obtenidos mediante

el módulo de Integración Temporal y el esquema integrado obtenido por el módulo de resolución se fijan los parámetros de refresco de los datos almacenados en el almacén. El Capítulo 7 explica con más detalle las tareas realizadas por el procesador.

2.7 Modelo de datos del almacén de datos corporativo

El punto de partida para la construcción del almacén de datos es el *modelo de datos* [Inmo02]. Sin un modelo de datos es muy complicado intentar organizar la estructura y contenido de los datos en el almacén de datos. Sin embargo, las técnicas clases de modelado no hacen distinción entre los entornos operacionales y decisionales. Las técnicas clásicas de modelado de datos simplemente intentan recopilar y sintetizar la información necesaria dentro de una organización. El resultado es el *modelo de datos corporativo*. El mejor punto de partida para la construcción del almacén de datos es el modelo de datos corporativo de la organización. Sin embargo, hay que hacer algunos cambios en dicho modelo, con el objeto de que esté listo para ser utilizado en la construcción del almacén.

Los almacenes de datos se construyen para una amplia variedad de aplicaciones y usuarios tales como sistemas de clientes, sistemas de marketing, sistemas de control de calidad, etc. Debido a la diversidad de aplicaciones y tipos de almacenes de datos, internamente cada uno de esos almacenes se centra alrededor de una estructura llamada *snapshot* o *instantánea*. Éstas, se crean cuando se produce algún evento, por ejemplo la finalización de una orden de pedido, una compra, atender a una llamada telefónica, la recepción de un encargo, etc. Estos ejemplos son actividades discretas que se producen como consecuencia de que algo ha ocurrido, y se toman notas del evento en el sistema direccional de la empresa. En general, las actividades discretas se dan de manera aleatoria. Otro tipo de snapshot son los que se producen en momentos predeterminados tales como final del día, final de la semana, final del mes, etc.

Una snapshot disparada por un evento tiene cuatro componentes básicos: una llave, una unidad de tiempo, datos primarios relacionados solo con la nave, y datos secundarios capturados como parte del proceso de snapshop y que no tienen relación directa con los datos primarios o la llave.

La llave puede ser única o no única y puede ser simple o compuesta de varios elementos. La llave identifica el registro y los datos primarios. Unidades de tiempo tales como año, mes, día, normalmente, aunque no siempre, se refieren al instante en el que el evento que es descrito por el snapshop ocurrió. Otras veces la unidad de tiempo se refiere al instante en que la captura tuvo lugar (transaction time en la fuente de datos). Algunas fuentes distinguen entre cuándo ocurrió el evento (valid time), y cuándo se captura la información sobre el evento (transaction time).

Los datos primarios están relacionados directamente con la llave. Como ejemplo supongamos que la llave identifica la venta de un producto, el elemento de tiempo

describe cuando se finalizó la venta, y los datos primarios describen cuando se venció el producto y a qué precio, condiciones de la venta, localización de la venta y las partes implicadas en la venta.

Los datos secundarios, si existen, contienen otra información capturada en el momento en el que se realiza la instantánea. Como ejemplo, relacionado con la venta del ejemplo anterior, podría incluirse información sobre el producto vendido (cuántas unidades quedan en stock en el momento de la venta).

Cuando se añade información secundaria, se puede deducir una relación entre la información primaria y la secundaria.

2.7.1 El modelo de datos corporativo

Para que pueda ser realizada la transformación del modelo de datos corporativo al modelo del almacén de datos, el primer modelo debe haber identificado y estructurado al menos lo siguiente:

- Las áreas más importantes de la organización
- Las relaciones entre las áreas
- La creación de un diagrama entidad relación
- Para cada una de dichas áreas: las claves, los atributos, los subtipos y los conectores de un área con la siguiente

El modelo de datos corporativo también puede incluir análisis sobre los procesos de la organización, tales como:

- Descomposición funcional
- Matrices de datos/procesos
- Diagramas de flujo de datos
- Diagramas de estado de transición
- Pseudocódigo, etc

El modelo de datos corporativo usualmente se divide en dos niveles: *alto nivel* y *medio nivel*. A alto nivel contiene las áreas más importantes de la organización y cómo ellas se relacionan. Y a medio nivel podemos encontrar las llaves, los atributos, los subtipos, los grupos de atributos y los conectores.

Cuando el modelo de datos corporativo abarca un ámbito muy amplio, se le llama modelo de datos empresarial. No todas las organizaciones detallan al mismo nivel el modelo de datos, pero en cualquier caso nos servirá como punto de partida para la construcción del almacén de datos.

2.7.2 Realizando la transformación

Existen unas actividades básicas necesarias para realizar la transformación entre los modelos. Son las siguientes:

- El borrado de los datos puramente operacionales
- La adición de un elemento de tiempo a la estructura del almacén
- La adición de datos derivados
- La transformación de las relaciones
- El ajuste de los diferentes niveles de granularidad
- La mezcla de datos provenientes de diferentes tablas
- La creación de matrices de datos, y
- La separación de los atributos de acuerdo a sus características de estabilidad

Pasamos a continuación a explicar cada uno de estos puntos.

2.7.2.1 El borrado de los datos puramente operacionales

En la Figura 11 podemos ver cómo los datos del modelo de datos corporativo se transforman al modelo de datos del almacén. Sin embargo algunos atributos son solo necesarios en el entorno operacional y por tanto, son eliminados antes de la construcción del modelo de datos del almacén. La eliminación de los datos operacionales es una decisión que debe tomar el diseñador del almacén. La cuestión a responder es si los datos serán usados en un futuro en los sistemas de ayuda a la decisión (DSS). Esta pregunta es muy complicada de responder, y finalmente hay que llegar a un compromiso entre almacenar todos los datos o sólo algunos. Para tomar la decisión hay que tener en cuenta el coste de almacenar gran volumen de datos que pueden o no ser utilizados en un futuro para los DSS.

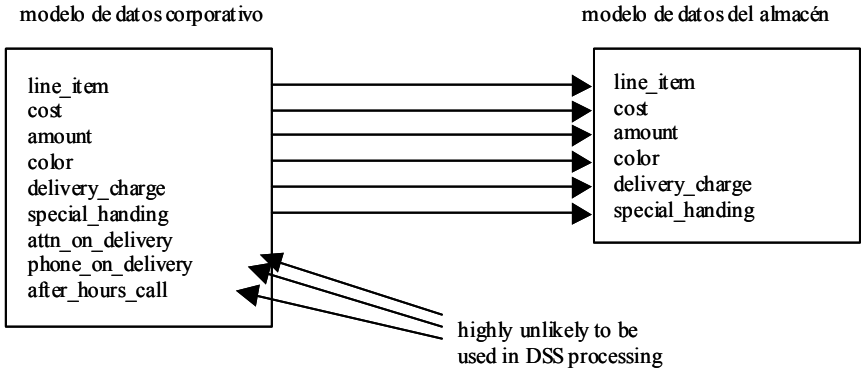

Figura 11. Eliminar los datos que no sean necesarios para el DW

2.7.2.2 Añadir un elemento de tiempo

La segunda modificación que debemos realizar es añadir un elemento de tiempo a la llave del almacén si aún no existe. La Figura 12 muestra cómo se añade un elemento de tiempo al registro de clientes. El modelo de datos corporativo tiene información sobre los clientes. Pero en el almacén las instantáneas de los datos necesitan tener marcas efectivas de tiempo. Por supuesto existen diferentes maneras a la hora de añadir un elemento de tiempo a la estructura del almacén de datos.

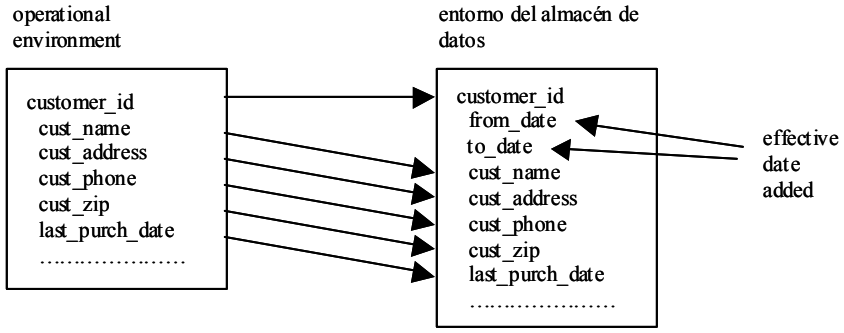

Figura 12. Añadir un elemento de tiempo

2.7.2.3 Añadir datos derivados

En la Figura 13 podemos apreciar cómo se añaden algunos datos derivados. Estos datos derivados suelen ser accedidos muy frecuentemente y solo son calculados una vez. Añadir datos derivados tiene sentido ya que reduce la cantidad de procesamiento requerido para acceder a los datos del almacén. Además, una vez calculados de manera adecuada, no habrá ninguna duda sobre la integridad de dichos datos.

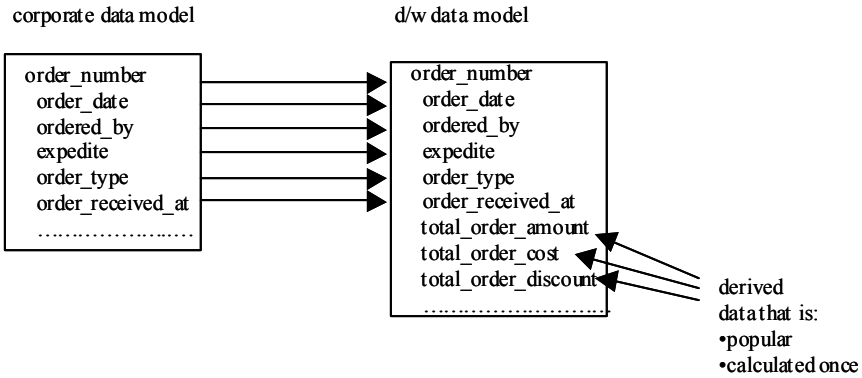

Figura 13. Añadir datos derivados

2.7.2.4 Transformación de las relaciones de los datos

Las relaciones de los datos encontradas en los modelos de datos clásicos están pensadas para entornos operacionales. Dichas relaciones asumen que hay una y sólo una regla de negocio representada en cada relación. Asumiendo que los datos son correctos en el momento de acceder a ellos (en el entorno operacional), la representación clásica de una relación es adecuada. Sin embargo, para el almacén de datos puede haber muchas reglas de negocio entre los datos de una tabla. Esto es debido a que los datos en el almacén se guardan a lo largo de un periodo de tiempo amplio y habrá muchas reglas de negocio a lo largo del tiempo. Por tanto, las relaciones entre las tablas en el almacén de datos se realizan a través de la creación de "artifacts".

Un artifact de una relación es simplemente la parte de la relación que es obvia y tangible en el instante de tiempo en que se realiza la instantánea de los datos del sistema operacional. El artifact puede incluir llaves foráneas y otra información de interés. Esta transformación es una de las más complejas a la hora de diseñar el almacén de datos. Un ejemplo simple lo podemos ver en laFigura 14, en la que hay una relación entre PART y SUPPLIER. En el ejemplo, cada PART tiene un SUPPLIER principal. Esta relación es la típica que se puede encontrar en el modelo de datos corporativo. Las relaciones de integridad indican que si un SUPPLIER se borra no deben existir PART que tengan a dicho SUPPLIER como principal suministrador. La relación representa algo que es activo y exacto en el momento de acceder a los datos. En la Figura 15 podemos ver cómo la instantánea de los datos se puede realizar y cómo la información sobre la relación se captura para ser representada en el almacén.

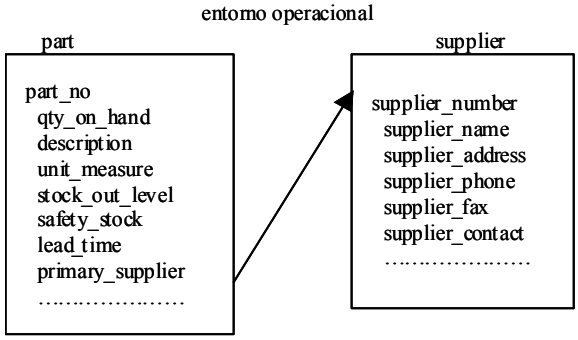

Figura 14. Relación en el sistema operacional

La tabla PART es creada periódicamente por ejemplo, al final de la semana, al final del mes, etc. A parte de otro tipo de información necesaria, uno de los datos que se captura en cada instantánea es el SUPPLIER principal. Esto es un ejemplo de cómo se captura una relación en el sistema operacional y es transformada a un artifact en el almacén. Hay que indicar que la relación es exacta en el momento en que se captura, pero ninguna otra implicación puede ser deducida de este hecho.

Las dos instantáneas mencionadas anteriormente tienen un serio inconveniente, son incompletas. Muestran sólo la relación existente en un instante de tiempo determinado, que es cuando se realiza la instantánea. Se pueden producir cambios de interés que nunca van a ser capturados. Por ejemplo, supongamos que la instantánea de PART se realiza cada semana, y que PART puede tener tres SUPPLIERs durante la semana.

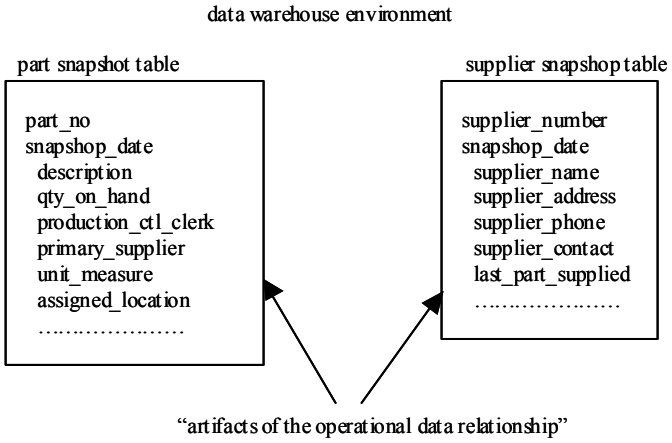

Figura 15. Relaciones en el DW

Si se realiza la instantánea al final de la semana, esta información nunca puede ser capturada. Otro ejemplo se podría dar si suponemos que las instantáneas de SUPPLIER se realizan mensualmente. Un SUPPLIER puede haber realizado quince pedidos diferentes de diferentes PARTs, pero la instantánea sólo mostrará el último pedido realizado.

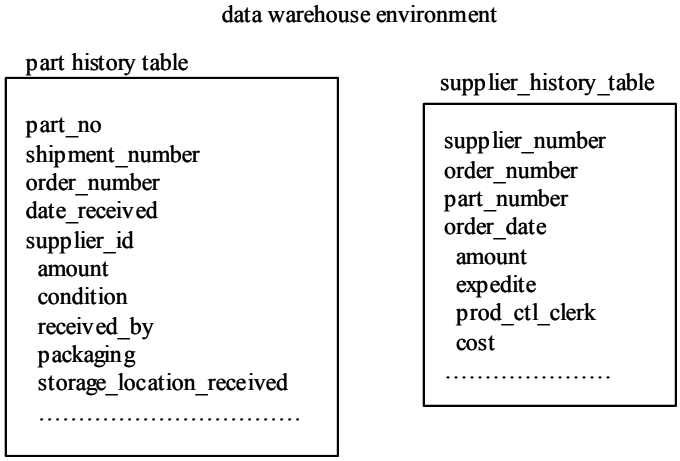

Figura 16. Otra opción en la que todas las actividades son capturadas

Las instantáneas son fáciles de realizar y son una parte esencial en la construcción del almacén de datos, pero tienen sus inconvenientes. Es deseable capturar la historia

completa de los datos, más que instantáneas sucesivas de los mismos. La Figura 16 muestra un ejemplo de datos históricos en el almacén.

En la tabla histórica PART, se recibe un pedido. Después de recibirlo, la información relevante se almacena, entre otras cosas el SUPPLIER de la PART. Esto es, por tanto, otra forma de artifact en el almacén de datos, asumiendo que todas las entregas tienen un registro histórico creado y por tanto el registro de las relaciones entre las dos tablas es completo. Lo mismo podemos decir para la tabla histórica SUPPLIER. El registro histórico se escribe siempre que un SUPPLIER recibe una orden. La totalidad de las órdenes recibidas por el SUPPLIER proporciona un registro completo de las relaciones.

2.7.2.5 Ajuste de los diferentes niveles de granularidad

Algunas veces los niveles de granularidad no cambian al pasar los datos del sistema operacional al almacén de datos, pero en otros casos los niveles de granularidad cambian y deben ser ajustados según las necesidades del almacén. La Figura 17 nos presenta un ejemplo del cambio de nivel de granularidad al pasar los datos al almacén. Podemos ver que en el entorno operacional hay una actividad que es recogida cada vez que se realiza un embarque. El diseñador del almacén de datos decide que hay que cambiar el nivel de granularidad y realizar dos sumarizaciones de los datos del embarque: la sumarización mensual del total de embarques y la sumarización de embarques por fuente/localización.

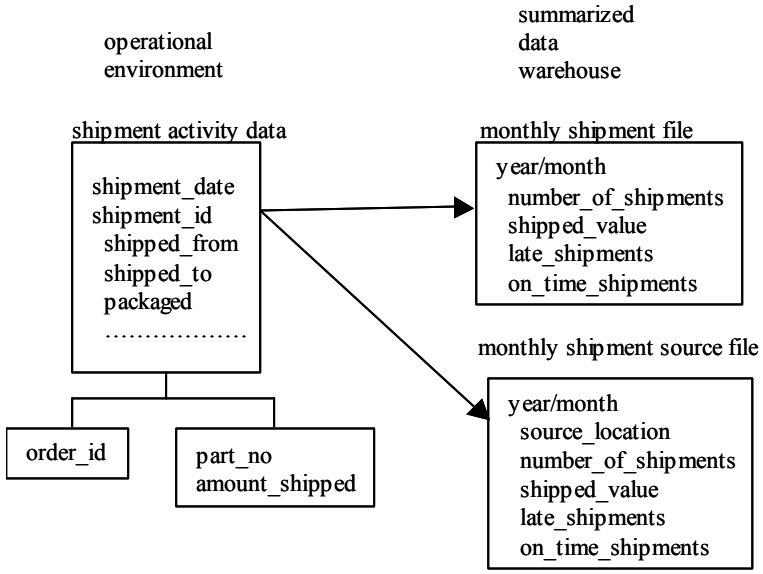

Figura 17. Granularidad del sistema operacional y del DW

2.7.2.6 Mezcla de datos provenientes de diferentes tablas

La siguiente transformación a considerar es cómo se debe realizar la mezcla de tablas del sistema operacional en tablas del almacén de datos. La Figura 18 ilustra la mezcla de tres

tablas normalizadas de un entorno de ingeniería/manufacturación, que son mezcladas al pasar los datos al almacén.

Las condiciones bajo las cuales tiene sentido realizar la mezcla de las tablas son: las tablas comparten una clave común, los datos de diferentes tablas se usan juntos frecuentemente y el patrón de inserciones es aproximadamente el mismo. Si alguna de esas condiciones no se da, no tiene sentido la mezcla de las tablas.

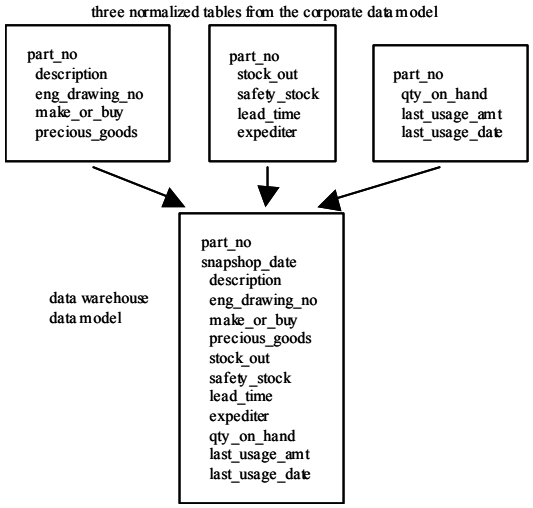

Figura 18. Mezclando las tablas del modelo de datos corporativo al modelo del DW

2.7.2.7 Creación de la matriz de datos

Los datos en el modelo de datos corporativo habitualmente están normalizados, esto significa que los grupos repetidos no se muestran como parte del modelo de datos. Pero bajo determinadas condiciones, el almacén de datos puede y debe contener grupos repetidos de datos. La Figura 19 muestra un ejemplo del modelo de datos del almacén que contiene matrices de datos. En el modelo de datos corporativo, como norma general los registros se crean mes a mes, pero al pasar los datos al almacén, se organizan de tal forma que cada mes del año es una ocurrencia de la matriz.

Podemos encontrar muchos beneficios para hacer esta estructuración de los datos. Una es que al no tener registros individuales para cada mes, se ahorra espacio de almacenamiento. La otra ventaja es la posibilidad de organizar todas las ocurrencias de los años en una sola localización física, y de esta forma aumentar la velocidad de acceso.

Solo bajo determinadas circunstancias tiene sentido crear matrices de datos en el modelo de datos del almacén. Las condiciones son:

- Cuando el número de ocurrencias de datos es predecible
- Cuando la ocurrencia de datos es relativamente pequeña (en términos de tamaño físico)

- Cuando las ocurrencias de los datos se usan frecuentemente juntos, y
- Cuando el patrón de inserción y borrado es estable.

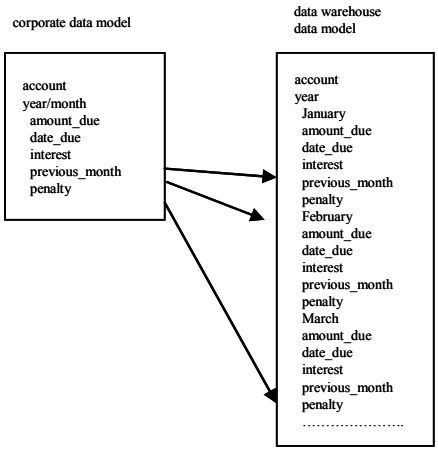

Figura 19. Crear un array de datos

2.7.2.8 Separación de atributos de acuerdo a sus características de estabilidad

La última transformación es la organización de los datos conforme a su propensión de cambio. El modelo de datos corporativo no hace distinción en el ratio de cambios de las variables contenidas en una tabla, pero el almacén de datos es muy sensible a la frecuencia de cambios de los datos que tiene almacenados.

La organización óptima implicaría que los datos en una tabla cambian todos "lentamente" y que los datos en otra tabla cambian todos "rápidamente", es decir, organizar los datos conforme a su "velocidad de cambio". La Figura 20 muestra los datos recogidos para un cliente, que están divididos en tres categorías: datos que nunca cambian, datos que cambian alguna vez y datos que cambian a menudo. Las estructuras del almacén de datos finalmente se pueden diseñar de forma que sean compatibles en términos de volatilidad con las tres categorías anteriores.

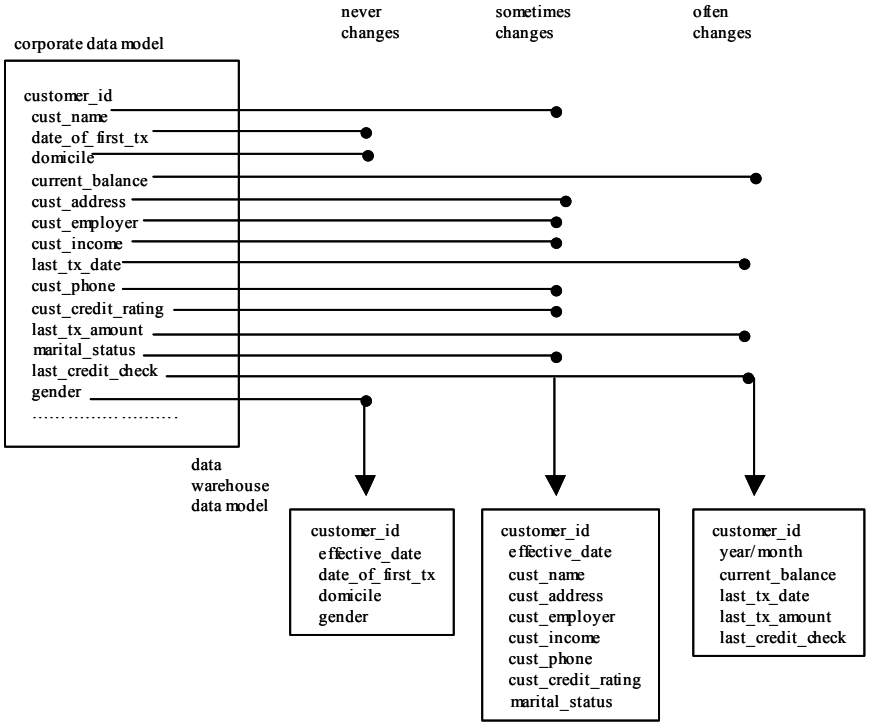

Figura 20. Dividir los datos conforme a su estabilidad

2.7.3 Orden de aplicación

El orden para aplicar los criterios de transformación debe ser el que hemos presentado anteriormente: en primer lugar la eliminación de los datos claramente operacionales, y en último lugar la agrupación de los datos de acuerdo a su estabilidad. Por supuesto, como en cualquier proceso de diseño, puede darse cierta iteración en los pasos. Sin embargo, como línea general, nos sirven los criterios antes presentados.

2.8 Fuentes de Datos y Métodos de extracción

2.8.1 Fuentes de datos

Las *fuentes de datos* pueden ser bases de datos operacionales, datos históricos (normalmente archivados en cintas), datos externos (por ejemplo de compañías de investigación de mercados o de internet), o información de almacenes de datos ya existentes. Además, las fuentes pueden residir en diferentes plataformas y contener información estructurada (como tablas u hojas de cálculo) o información no estructurada (como ficheros de texto plano, imágenes o información multimedia).

La *Extracción, Transformación y Carga* (ETL), son procesos dentro de la construcción del almacén de datos que implican: extraer datos de fuentes externas, adaptarlos a las necesidades del negocio y cargarlos en el almacén.

La primera parte del proceso ETL es *extraer* los datos desde los sistemas fuente. La mayoría de los proyectos de almacenes de datos tienen información integrada a partir de diferentes fuentes de datos. Cada fuente puede tener una forma diferente de organizar y formatear sus datos. El proceso de extracción suele convertir los datos en registros y columnas.

La fase de *transformación* aplica una serie de reglas o funciones a los datos previamente extraídos para derivar los datos que serán posteriormente cargados en el almacén.

Durante la fase de *carga*, los datos son guardados en el almacén. Dependiendo de los requerimientos de la organización, el proceso puede variar desde simplemente sobrescribir información antigua con nuevos datos, hasta sistemas más complejos que mantienen la historia completa de todos los cambios producidos en los datos.

2.8.2 Captura de datos

Los almacenes de datos describen la evolución histórica de una organización, y el marcado temporal de los datos nos permite mantener dichos datos históricos. Cuando hablamos de datos temporales en el ámbito de los almacenes de datos es necesario entender cómo se refleja el tiempo en las fuentes de datos, cómo este se relaciona con la estructura de los datos y cómo un cambio de estado afecta a los datos existentes. Se han propuesto varias aproximaciones [Bruc02], [Jens92]:

- *Transient data*: las alteraciones y borrados de registros existentes físicamente destruyen el contenido de los datos anteriores.
- *Semi-periodic data*: normalmente encontrados en datos de tiempo real de sistemas operacionales donde los estados previos son importantes. Sin embargo, casi todos los sistemas operacionales solamente retienen una pequeña historia de los cambios de los datos, principalmente debido a restricciones de rendimiento y almacenamiento.
- *Periodic data*: cuando un registro se añade a la base de datos nunca se borra físicamente ni puede ser modificado. Se añaden nuevos registros que reflejan actualizaciones y borrado de los datos. Por tanto, los datos periódicos contienen un registro completo de todos los cambios producidos.
- *Snapshot data*: es una vista estable de los datos, tal y como existen en un instante de tiempo dado.

La *captura* es un componente de la replicación de datos que interactúa con las fuentes de datos para obtener una copia de algunos o de todos los datos contenidos en ella o un registro de cualquiera de los cambios [Devl97]. En general, no necesitaremos todos los

datos disponibles en la fuente. Aunque tengamos la opción de capturar todos los datos y deshacernos de los que no nos interesen, es más eficiente capturar solo el subconjunto de datos necesarios. La captura de dicho subconjunto, sin ningún tipo de referencia temporal en la fuente, se denomina *captura estática*. Cuando las fuentes de datos cambian con el paso del tiempo, es necesario capturar la historia de esos cambios. Algunas veces realizar una captura estática repetidas veces es suficiente, sin embargo la mayoría de las veces necesitamos capturar los cambios actuales que han ocurrido en la fuente. La necesidad de transformar datos transitorios o semiperiódicos en datos periódicos, es el principal requerimiento que nos lleva a realizar otro tipo de captura denominada *captura incremental*.

La *captura estática* esencialmente toma una instantánea de la fuente de datos en un punto en el tiempo. Esta instantánea puede contener todos los datos encontrados en la fuente, pero realmente solo contiene un subconjunto de los datos. La captura estática tiene lugar la primera vez que un conjunto de datos de un sistema operacional va a ser cargado en el almacén de datos. En este caso el sistema operacional mantiene una historia completa de los datos, y el volumen de datos suele ser pequeño.

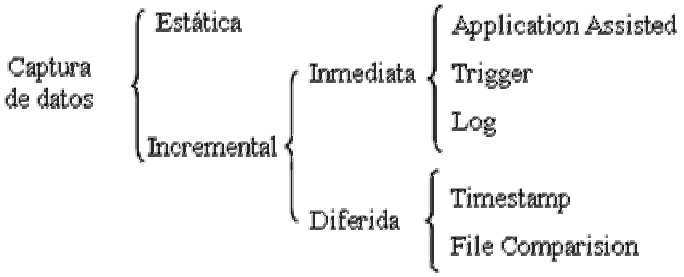

Figura 21. Métodos de captura de datos

La *captura incremental* es el método para capturar el registro de cambios ocurrido en una fuente de datos. Este método supone que la mayoría de los datos tienen una dependencia temporal, y por tanto se requiere una aproximación para manejar eficientemente esta característica. Como el volumen de cambios en un conjunto de datos es casi siempre más pequeño que el volumen total, una captura incremental de los cambios en los datos es mucho más eficiente que una captura estática.

La *captura diferida* tiene lugar en instantes predefinidos, más que cuando los datos cambian. En datos periódicos, esto implica que se puede obtener un registro completo de todos los cambios producidos en la fuente. En datos transitorios y semiperiódicos, puede implicar bajo determinadas circunstancias que no se puedan obtener todos los cambios de los datos producidos en la fuente, debido a que entre dos capturas predefinidas pueden haberse producido múltiples borrados y actualizaciones de los datos.

La captura incremental no es una tarea fácil y se divide en cinco técnicas diferentes, cada una de las cuales tiene sus ventajas e inconvenientes. Las tres primeras se denominan *captura inmediata*, en la que los cambios que se producen en la fuente se capturan

justamente después de haberse producido y que nos garantiza que todos los cambios producidos en la fuente pueden ser capturados independientemente de que los datos sean transitorios, semiperiódicos o periódicos. Los tres tipos de captura inmediata son:

- *Application-assisted*: es necesario cambiar la aplicación del sistema operacional para que los cambios que se produzcan en los datos se almacenen de forma permanente
- *Triggered*: depende de que el administrador de la base de datos almacene los cambios de los datos de manera permanente. Hay dos posibles formas de manejar los cambios producidos mediante este tipo de captura:
  - *Triggered Directo*: el administrador de la base de datos es el que debe encargase de diseñar el trigger de forma que actualice la información de las aplicaciones que requieran conocer los cambios producidos.
  - *Triggered Delta*: en este caso, los cambios producidos se almacenan en un fichero temporalmente, de forma que la aplicación que los requiera debe consultar este fichero antes de que sea eliminado.
- *Log/journal*: depende de que el administrador de los log/journal de la base de datos almacene los cambios de forma permanente

En algunos entornos no es posible utilizar las técnicas anteriores de captura incremental, pero podemos utilizar estrategias de *captura diferida*, tales como:

- *Timestamp-based*: selecciona los datos cambiados basándose en marcas de tiempo proporcionadas por el programa de aplicación que mantiene los datos
- *File comparison*: compara versiones de los datos para detectar cambios en los mismos

2.8.3 Clasificación de los métodos de extracción

Clasificar los métodos de extracción con los que se capturan los datos de las fuentes nos será de utilidad para poder ofrecer una interface de cada una de ellas y así poder "exportar" las características de las mismas. Éstas serán utilizadas para diseñar el esquema del almacén de datos y para definir el proceso de refresco a realizar.

Dependiendo de la cantidad de datos que se capturen de la fuente Inmon [Inmo96] propone la siguiente clasificación:

- *Timestamp-based*: se capturan los datos que previamente han sido señalados con una marca de tiempo. En este caso es necesario que las aplicaciones en las fuentes de datos realicen el marcado cuando se produzca un cambio o una actualización en los datos.

- *Delta-file*: en este caso la fuente proporciona un fichero delta con los cambios producidos desde la última captura de datos. El procesamiento es muy eficiente pero la mayoría de las fuentes no disponen de esta facilidad.
- *Log-file*: el fichero de log contiene la misma información que el fichero delta y adicionalmente otra que el sistema donde se genera estima necesaria, como por ejemplo, trazas para la recuperación del sistema. El fichero log suele tener un formato específico lo que conlleva un problema añadido: sería necesario desarrollar una interface para ese fichero, a parte de capturar los datos que necesitemos del mismo.
- *Application-assisted capture*: modificar los programas de aplicación existentes para que generen los datos que deben ser capturados. Esto en la mayoría de los casos es inviable dado que sería necesario modificar código fuente de aplicaciones en algunos casos obsoletas o programadas en algún lenguaje poco difundido.
- *File comparison*: consiste en realizar instantáneas sucesivas de la fuente en cuestión cada vez que se va a realizar la captura de los datos de la misma. Las instantáneas sucesivas en el tiempo se comparan para averiguar que actividad se ha producido en la fuente y que datos nos interesa capturar.

Widom [Wido95] distingue entre las siguientes fuentes de datos:

- *Fuentes cooperativas*: disponen de triggers u otros mecanismos activos, para que los cambios de interés puedan ser programados e se informe automáticamente de los mismos.
- *Fuentes log*: mantienen un fichero log que puede ser inspeccionado para detectar los cambios producidos en las fuentes y extraer los datos que nos interesen.
- *Fuentes interrogables*: permiten a los monitores consultar los datos de las fuentes, de tal forma que se puede realizar un muestreo periódico para detectar posibles cambios.
- *Fuentes snapshot*: se realiza un volcado periódico del contenido total de la fuente y se pueden detectar cambios de interés comparando un volcado con otro.

En [Vavo99] se realiza la siguiente descripción y agrupación de los métodos de extracción:

- *Replication-based monitoring*: se utilizan los servicios proporcionados por un DBMS comercial para detectar los cambios producidos en la fuente. Herramientas como IBM's Data Propagator y Sybase's Replication Server proporcionan mecanismos para propagar actualizaciones en tablas base.
- *Log-based monitoring*: se supone que el sistema fuente es una base de datos con un log asociado, el cual contiene información sobre las transacciones realizadas. Cuando se inicia el proceso de refresco, el monitor inspecciona el fichero log y extrae la información relevante que ha ocurrido desde el último refresco.
- *Trigger-based monitoring*: son fuentes que soportan mecanismos activos como triggers, y en las que se pueden definir eventos de interés que serán disparados en

el momento adecuado. En este caso, la acción de un trigger graba información de interés (actualización de entidades y atributos, la clase de operación de actualización y una marca de tiempo) dentro de una tabla auxiliar. Después del refresco del almacén de datos, se puede eliminar el contenido de dicha tabla.

- *Application-assisted extraction*: se utiliza en fuentes de datos que no son DBMS, y consiste en cambiar los programas de aplicación o implementar unos nuevos, de tal forma que notifiquen sobre los cambios de interés que se han producido en la fuente. La creación de snapshop de información relevante y la comparación (registro a registro) con una versión previa de dicha snapshop, también se utiliza para este tipo de fuentes. Otra opción es cambiar los programas de aplicación para realizar una marca de tiempo en los datos que han cambiado. El programa de monitorización consulta periódicamente la fuente y selecciona los datos con una marca de tiempo mayor que la del refresco anterior.

En [Jark00] la clasificación que el autor propone es:

- *Specific*: cada fuente es un caso particular, y no es posible usar un método general. Ejemplos de este tipo de fuentes son los sistemas legados, debiéndose utilizar un método específico dependiendo de cada fuente. Por ejemplo, algunos sistemas legados son capaces de generar ficheros delta que describen sus acciones, en otras ocasiones los sistemas legados realizan una marca de tiempo en sus registros.
- *Internal Action*: similar a las fuentes callback, excepto que en este caso no es posible definir un trigger. Este método requiere definir trigger en la fuente y crear relaciones auxiliares llamadas tablas delta.
- *Logged*: se dispone de un fichero log donde están registradas todas las acciones de interés. Para este tipo de fuentes, los cambios se detectan consultando periódicamente el log de la fuente y analizándolo. Sin embargo, el log no es de fácil acceso (en ocasiones se requiere autorización del administrador de la base) y la interpretación de los registros suele ser complicada, ya que no están en un formato estándar.
- *Replicated*: en este caso se utiliza el servicio de replicación del sistema operacional. Para detectar los cambios de interés que se han producido en la fuente se analizan los mensajes que se envían por el sistema de replicación. Este servicio suele ser proporcionado por las marcas más relevantes de BD.
- *Callback*: se dispone de triggers u otras capacidades activas que permiten detectar automáticamente los cambios de interés y notificarlos. Es necesario tener acceso al sistema de la fuente de datos para definir los trigger.
- *Snapshot*: se proporciona el contenido completo de la fuente en una instantánea de la misma, sin tener ninguna capacidad para seleccionar los datos de interés. Los cambios producidos se hacen comparando dos instantáneas sucesivas de la fuente y detectando los cambios.

- *Queryable*: se ofrece una interface para realizar consultas a la fuente. Hay que hacer consultas periódicas a la fuente para detectar si el dato en el que estamos interesados ha cambiado desde la última consulta. Ejemplo de este tipo de fuentes son las BD relacionales.

Dependiendo del grado de *capacidad activa* de la fuente podemos encontrar otra clasificación en [Engs00]. Se dividen según la capacidad de enviar información referente a los cambios producidos desde la fuente al sistema de integración. En este caso las fuentes sólo pueden ser bases de datos. Se dividen en:

- *Cooperación suficiente*: es capaz de enviar deltas (cambios incrementales) correspondientes a todas las actualizaciones desde la transmisión previa. Las fuentes de datos disponen de trigger basados en eventos físicos o en cambios de estado.
- *Cooperación restringida*: en este caso no puede enviar deltas, pero dispone de trigger basados en algún evento físico (ejecución de métodos o finalización de transacciones), y la habilidad para enviar mensajes simples al sistema de integración.
- *Cooperación nula*: la fuente no tiene capacidad de enviar ningún tipo de información. En este caso, hay que consultar la fuente periódicamente y realizar una actualización parcial o completa de los datos.

Otra forma de clasificar las fuentes es en *estructuradas*, *semi-estructuradas* o *no-estructuradas*. En las estructuradas se incluyen las bases de datos, en las semi-estructuradas los ficheros HTML/XML y en las no-estructuradas los ficheros de texto, imagen, voz, etc.

Algunas fuentes y los métodos de extracción asociados para obtener los datos podrán proporcionarnos determinados parámetros temporales y otras no. Dependiendo de si dichas fuentes manejan *Transaction Time* (instante de tiempo que indica que un hecho esta disponible en las fuentes de datos,TT) y/o *Valid Time* (intervalo de tiempo en el que un hecho es cierto en la realidad que se modela, VT) podremos incorporar dichos parámetros al esquema de las fuentes y posteriormente, si fuera de interés, al esquema del almacén de datos.

En el caso del *Transaction Time*:

- Si el TT se puede obtener de la fuente de datos lo incorporaríamos tal cual al almacén de datos. El dato extraído de la fuente se marcaría y almacenaría con el TT de la propia fuente de datos. Es decir, $TT_{dato} = TT_{fuente}$
- Si, ni la fuente de datos ni el método de extracción pueden proporcionar el TT, éste puede ser gestionado por el DBMS del almacén datos en el momento en se actualice graba el dato de la fuente en el almacén. El TT del dato sería el TT que proporciona el DBMS del almacén al grabar el dato. Es decir, $TT_{dato} = TT_{ad}$

En el caso del *Valid Time*:

- Si el VT se puede obtener de la fuente de datos lo incorporaríamos tal cual al almacén de datos. El dato extraído de la fuente se marcaría y almacenaría con el VT_{inicio} de la propia fuente de datos. El VT_{end} permanecería igual a un valor genérico *Now* [Clif00] hasta que se detectara el cambio correspondiente en la fuente y el VT_{end} se igualara al nuevo valor.
- Si, ni la fuente de datos ni el método de extracción pueden proporcionar el VT, se puede aproximar el VT_{inicio} con el TT_{ad} (instante en que se graba el dato en el DBMS del almacén). El VT_{end} permanecería igual a un valor genérico *Now* [Clif00] hasta que se detectara el cambio correspondiente en la fuente y el VT_{end} se igualara al nuevo valor TT_{ad}

Tipos de fuentes	TT	VT
Delta-file (Cooperación suficiente)	S	N
Log-file, Fuentes log (Cooperación suficiente)	S	N
Timestamp-based, Fuentes cooperativas (Cooperación suficiente)	S	N
Application-assisted capture (Cooperación suficiente).	S	S
File comparison y Fuentes snapshot (Cooperación restringida)	S	S
Fuentes interrogables (Cooperación nula)	N	N

Tabla 1. TT y VT según el tipo de Fuentes de datos

Para cada una de las fuentes descritas anteriormente podemos ver si pueden proporcionarnos el TT y VT en la Tabla 1.

2.8.3.1 Clasificación agrupada

Siguiendo la clasificación presentada en la sección 2.8 y en concreto en la Figura 21 podemos agrupar los métodos descritos por los diferentes autores en cinco categorías. Algunos métodos pueden estar en más de una categoría dado que de la definición propuesta por los autores así se deduce. En concreto, tenemos los siguientes grupos (Tabla 2):

- Grupo 1: Application Assisted
- Grupo 2: Trigger
- Grupo 3: Log
- Grupo 4: Timestamp
- Grupo 5: File Comparison

	[Inmon, 96]	[Widom, 95]	[Sirius]	[Jarke et al, 03]	[Engström et al., 00]
G1	Application-assisted	-----	Application-assisted	-----	Cooperación restringida
G2	Delta-file	Cooperativas	Replication / Trigger	Specific / Internal Action/ Callback	Cooperación suficiente/restringida
G3	Log-file	Log	Log	Logged / Replicated	Cooperación suficiente/restringida
G4	Timestamp			Specific	Cooperación restringida
G5	File comparison	Snapshot/ Interrogables	Application-assisted	Snapshot / Queryable	Cooperación nula

Tabla 2. Métodos de extracción agrupados por su similitud

Esta clasificación es la que será utilizada en los siguientes capítulos de la tesis. Pasamos a continuación a describirlos brevemente. En la Figura 22 vemos las diferencias entre los diferentes métodos.

2.8.3.1.1 Grupo 1: Application Assisted

Los métodos de extracción que están basados en modificar los programas de aplicación existentes para que generen los datos que deben ser capturados son los incluidos en este grupo. Una vez modificados los programas de aplicación se puede producir un fichero delta que es el que será utilizado para detectar los cambios. En la mayoría de las ocasiones es muy complicado utilizar este método dado que sería necesario modificar código fuente de aplicaciones en algunos casos obsoletas o programadas en algún lenguaje poco difundido.

2.8.3.1.2 Grupo 2: Trigger

Este grupo los forman los métodos basados en trigger u otros mecanismos activos que permiten que los cambios de interés puedan ser programados y se informe automáticamente de los mismos. Se incluyen los métodos de extracción que son capaces de generar un fichero delta donde se van registrando los cambios que se producen en la fuente de datos. El procesamiento es muy eficiente pero la mayoría de las fuentes no disponen de esta facilidad. En algunas fuentes existe la posibilidad de definir trigger y crear relaciones auxiliares llamadas tablas delta.

2.8.3.1.3 Grupo 3: Log

En este caso estarían incluidos los métodos que manejan un fichero log donde se graban los cambios producidos en las fuentes. El fichero de log contiene la misma información que el fichero delta y adicionalmente otra que el sistema donde se genera estima necesaria,

como por ejemplo, trazas para la recuperación del sistema. Suele tener un formato específico lo que conlleva un problema añadido: sería necesario desarrollar una interface para ese fichero, aparte de capturar los datos que necesitemos del mismo.

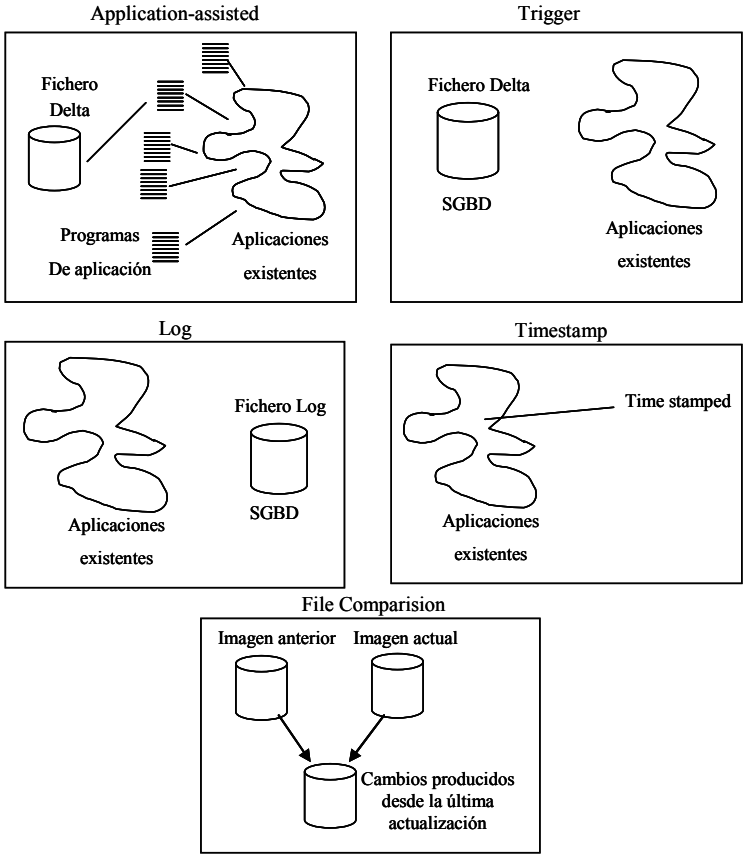

Figura 22. Métodos de extracción

2.8.3.1.4 Grupo 4: Timestamp

En este grupo se agrupan los métodos de extracción basados en la modificación de las aplicaciones en las fuentes de datos para que realicen el marcado cuando se produzca un cambio o una actualización en los datos.

2.8.3.1.5 Grupo 5: File Comparision

En este grupo incluimos los métodos que se basan en consultar las fuentes para realizar instantáneas cada vez que se va a realizar la captura de los datos de la misma. Las instantáneas sucesivas en el tiempo se comparan para averiguar que actividad se ha producido en la fuente y que datos nos interesa capturar.

2.9 Conclusiones

La integración de información no es una tarea sencilla e implica superar varias fases para poder realizarla con éxito. Es necesario disponer de una arquitectura adecuada que nos sirva como referencia para desarrollar una metodología de integración adecuada a las necesidades del marco del problema a resolver. En el ámbito de los almacenes de datos además es necesario tener en cuenta otros factores tales como la dimensión temporal de los datos, de las fuentes de datos y de los métodos extracción de éstos. En este capítulo se han revisado algunas propuestas existentes para la integración de datos en general y para el caso de los almacenes de datos en particular. Una de las partes de la integración es la construcción del sistema federado. También se ha presentado nuestro marco de referencia y se ha propuesto una arquitectura funcional para resolver el problema de la integración de las propiedades temporales de los datos. Esta arquitectura también se he presentado de forma más detallada describiendo cada uno de los módulos. Es necesario que el almacén de datos corporativo disponga de un modelo de datos ajustado a las necesidades del almacén, por tanto hemos presentado cuáles deben ser dichas características. Por último se han presentado los métodos de extracción propuestos por diferentes autores y se ha propuesto una clasificación agrupada de los mismos.

Capítulo 3

MODELOS DE DATOS ORIENTADOS A OBJETOS Y MODELOS TEMPORALES

En algunas áreas se utilizan conceptos que pueden ser de interés en el campo de almacenes de datos para resolver problemas relacionados con: el enriquecimiento de los metadatos temporales de las fuentes de datos, la extracción de los datos, el proceso de refresco, y la definición del esquema del almacén. En este capítulo se realiza una revisión de los modelos temporales más relevantes, centrándonos en los que proponen una extensión temporal de algún modelo de datos ya existente como, por ejemplo, el modelo de objetos ODMG, siendo éste modelo el que nosotros utilizaremos como modelo canónico. Basándonos en los modelos propuestos por otros autores y en los elementos temporales que necesitamos definiremos nuestro modelo de tiempo.

Este capítulo se estructurado como sigue: En la sección 3.1 se ilustrarán brevemente los elementos temporales utilizados en otras áreas. En la sección 3.2 se revisarán algunos modelos temporales existentes. En la sección 3.3 se propondrá el modelo de tiempo que servirá de base a los capítulos posteriores. En la sección 3.4 se presentará el standard ODMG 3.0.

3.1 Elementos temporales

Existen campos en los que claramente el factor tiempo es muy importante. Consideremos entornos en los que el estado del sistema cambia rápida y continuamente como, por ejemplo, el control del tráfico aéreo, el manejo de redes o los procesos de control. Las bases de datos se han utilizado ampliamente para almacenar los datos que dichas aplicaciones necesitaban y reflejaban los datos que indicaban el estado del entorno con tanto fiabilidad como era posible. Esta tarea no era fácil debido al continuo cambio del entorno, que daba lugar a continuas actualizaciones en la base de datos. Además, el sistema debía atender la petición de transacciones procedentes de la aplicación en cuestión. Teniendo en cuenta estas circunstancias, en muchas ocasiones había que llegar a

un compromiso y decidir si se dedicaba más esfuerzos a actualizar los datos, siendo el estado más consistente, o bien se primaba atender las transacciones. Si se hacía esto último podía ocurrir que los datos con los que se calculaba el resultado de la transacción no reflejaran el estado real del entorno.

Un Sistema de *Bases de Datos de Tiempo-Real* (SBDTR) debe facilitar transacciones y consultas en tiempo-real con restricciones explícitas de tiempo, tales como el tiempo límite de ejecución de la transacción [Kao95], [Lin94]. Las *Bases de Datos Temporales* (BDT) están diseñadas para capturar información variante en el tiempo [Rama93], a la vez que proporcionan un marco para mantener la historia de los cambios producidos en una fuente de datos. Existen aplicaciones en las que ambos requerimientos son necesarios, por ejemplo, muchas transacciones en las BDTR usan datos anteriores para calcular su resultado. Ambas bases de datos se complementan una a otra [Stan99].

Los esfuerzos en el campo de las BDTR se han centrado, fundamentalmente, en el desarrollo de algoritmos de planificación y de control de concurrencia, orientados, principalmente, a mejorar el rendimiento de las transacciones, pero ninguno se ha centrado en la consistencia temporal de los datos. Los trabajos sobre BDT se han concentrado en el enriquecimiento de modelos tradicionales (relacional y orientado a objetos) de bases de datos con características de tiempo y en el de lenguajes de consulta como el SQL, sin preocuparse de relacionar la dimensión temporal con un contexto de tiempo-real.

Un sistema de tiempo-real generalmente se compone de un sistema de control y de un sistema controlado. Por ejemplo, el sistema controlado podría ser un robot y el sistema de control un programa que recibe el estado del robot y le envía las órdenes oportunas. Es necesario, por tanto, mantener la consistencia entre el estado actual del entorno y el estado almacenado en la base de datos. Este es el concepto de consistencia temporal [Rama93] que está formado por dos componentes:

- *consistencia absoluta*, entre el estado real del entorno y cómo se refleja este estado en la base de datos.
- *consistencia relativa*, entre los datos usados para derivar otros datos.

Otro concepto a tener en cuenta es el tipo de periodicidad de las transacciones, el cual se deriva a partir de los requerimientos de la aplicación. Desde ese punto de vista, las transacciones se clasifican en [Rama93]:

- *Periódicas*: Se producen a intervalos fijos de tiempo, de tal forma que se puede saber en qué instante va a empezar a ejecutarse una transacción.
- *Aperiódicas*: Son el resultado de la interacción entre el entorno y el sistema que controla al entorno; se pueden producir en cualquier instante de tiempo.

Otros conceptos relacionados con el tiempo son [Snod86]:

- *Intervalo válido*: Intervalo de tiempo en el cual un dato es válido; la vida de un objeto especifica el intervalo de tiempo entre su nacimiento y su muerte.
- *Vida del intervalo válido*: Intervalo en el cual un determinado objeto existe en el sistema. Por ejemplo, el estado de un sensor puede marcarse con un valor que representa el instante de tiempo en el que fue recogido el dato por el hardware. Una vez almacenado en la base de datos, el dato puede caducar y sería necesario actualizarlo dentro de un determinado periodo de tiempo (intervalo válido).
- *Tiempo de transacción* (TT): Momento en el que un hecho se graba en la fuente de datos.
- *Tiempo de validez* (VT): Intervalo de tiempo en el que un hecho es cierto en la realidad que se modela. Un determinado evento genera un intervalo de validez y un intervalo de transacción.

Existen numerosos trabajos sobre BDTR y BDT de forma aislada, pero muy pocos en los que se relacionen ambas de forma directa [Rama93],[Data96]. En [Adel95] se presenta un sistema que proporciona los servicios básicos de una base de datos (SQL, recuperación, índices) con características de tiempo-real (deadline de transacciones). Es un sistema de tiempo-real blando donde los recursos necesarios no se saben con antelación. El sistema intenta minimizar el número de deadline violadas y considera sólo la vista actual, en oposición a las vistas históricas. Por tanto, el sistema sólo necesita almacenar la última versión de cada dato para poder operar, y no tiene en cuenta la consistencia temporal. Esto no es real en la mayoría de los casos, siendo necesario almacenar vistas históricas de los datos.

En resumen, las BDT se pueden utilizar para mantener los cambios producidos en los objetos del mundo real, pero, dado que el valor de los diferentes objetos permanece en la base de datos aunque su periodo de validez finalice, estas bases de datos realmente no están actualizadas con los cambios producidos en el entorno y no reflejan el estado actual. Las BDTR necesitan a las BDT para acceder a los datos necesarios para llevar a cabo un determinado cálculo basado en estados anteriores[Stan99].

Como se ha podido apreciar anteriormente, la actualización del almacén de datos debe llevarse a cabo en determinados momentos y bajo determinados criterios. Si disponemos de un modelo que represente las características temporales de las distintas fuentes, el diseñador del almacén de datos podrá desarrollar su trabajo de una manera más fácil y exacta. Algunas de las características de las fuentes de datos se corresponden con conceptos de tiempo utilizados en las bases de datos de tiempo-real y en las bases de datos temporales [Stan99].

Las *bases de datos de tiempo real* y las *bases de datos temporales* utilizan muchos conceptos que pueden ser aplicados en la construcción y mantenimiento del almacén de datos. Características como temporalidad de los datos (datos temporales y datos no temporales), frecuencia de acceso a los datos (muy frecuentes, pocos frecuentes), consistencia temporal

de los datos (relativa, absoluta), pueden utilizarse para definir las características de una fuente de datos añadiéndole información temporal. Esto facilitaría tanto la definición del modelo de datos del almacén con funcionalidades de tiempo, como la carga en sí misma.

3.2 Revisión de modelos

A continuación se resumen las principales características de modelos que, basándose en otros ya definidos, los han extendido para añadirle propiedades y características temporales.

3.2.1 T_CHIMERA

El modelo de datos T_Chimera [Bert96] es una extensión temporal del modelo de datos orientado a objetos Chimera. Por cada uno de los tipos de datos que provee Chimera se define un nuevo tipo temporal. Además, en T_Chimera define un nuevo tipo básico denominado time.

Una clase en T_Chimera consiste en dos componentes: la marca y la implementación. La marca de una clase contiene toda la información para poder hacer uso de la clase y sus instancias, información sobre la identificación y su tipo, mientras que la implementación provee de la implementación de la marca. Una clase es estática si todos sus atributos son estáticos. Si alguno de sus atributos puede modificar su valor a lo largo del tiempo se considera que la clase es histórica. Puesto que los objetos pertenecientes a una clase pueden variar a lo largo del tiempo, cada clase mantiene también una historia de todas sus instancias o miembros de la clase.

Cada clase queda definida mediante una tupla de la forma (c, type, lifspan, attr, meth, history, mc), donde:

- c es un identificador
- type indica si se trata de una clase histórica o estática
- lifespan es el intervalo en el que es válida
- attr contiene información sobre los atributos de la clase
- meth contiene información acerca de los métodos de la clase
- history es un valor que contiene pares de valores de la forma (nombre de atributo, valor), una lista de identificadores de miembros de la clase y de instancias asociados a cada instante del lifespan de la clase
- mc es el identificador de la metaclase de la que es instancia

Cada objeto se define mediante una tupla de la forma (id, lifespan, v, class-history) donde:

- id es el identificador del objeto
- lifespan el periodo de vida del objeto.

- v es un valor que contiene pares de valores de la forma (nombre de atributo, valor) para cada instante del periodo de vida.
- class-history es un conjunto de pares de valores que indican la clase más específica a la que ha pertenecido el objeto en cada momento

Los valores de las propiedades temporales de un objeto deben mantenerse consistentes con respecto a la clase a la que pertenezca a lo largo del tiempo, incluso si el objeto pasa a ser una instancia de otra clase que no tenga tal propiedad temporal. Un objeto es históricamente consistente si dado un instante perteneciente a su periodo de vida, el valor de sus atributos son valores legales del tipo de objeto al que pertenece en ese mismo instante. Un objeto es consistente si para cada instante de su periodo de vida sus atributos tienen un valor legal con respecto a la clase a la que pertenece en ese instante.

Mediante relaciones de herencia se permite refinar un atributo estático en uno temporal (sobre el mismo dominio o en uno más específico), pero no a al contrario. Además, para asegurar la sustitubilidad, se introduce el concepto de función de coerción, ya que el valor de un atributo temporal, que es realmente una función en el dominio del tiempo, no puede ser sustituido por un valor no temporal, que no es una función. La función de coerción utiliza el valor del último de los instantes conocidos como valor del atributo estático y descarta su historia.

3.2.2 T-ODMG

El objetivo del modelo T_ODMG, propuesto en [Bert98], es incorporar características temporales al modelo estándar de datos orientado a objetos, ODMG. Se basa en el modelo de datos formal T_Chimera, propuesto por los mismos autores en [Bert96].

Básicamente, el conjunto de tipos de datos que provee ODMG ha sido completado con otro conjunto de tipos de datos temporales que permiten el manejo de manera uniforme tanto de los objetos temporales como los que no lo son. A continuación se añade la dimensión temporal tanto a los atributos de los objetos como a las relaciones entre estos últimos.

Una de las características relevantes que posee T_ODMG es que ofrece soporte para propiedades de objeto temporales, inmutables y estáticas. Una propiedad temporal es una propiedad cuyo valor puede cambiar a lo largo del tiempo, y cuyo valor en los diferentes instantes queda almacenado en la base de datos. Una propiedad inmutable es aquella cuyo valor no puede cambiar durante el periodo de vida del objeto, mientras que una propiedad estática es una propiedad cuyo valor puede cambiar a lo largo del tiempo, pero sus valores pasados no son significativos, y son por tanto descartados.

El tipo de datos estructurado TimeStamp proporciona la granularidad temporal del modelo. Representa un instante t y es, por tanto, representado como una pareja de valores <fecha, hora>. En [Bert03] le añade la posibilidad de soportar múltiples granularidades

temporales en un mismo esquema. A partir de entonces se debe especificar la granularidad de cada uno de los tipos temporales que se definan.

La función de coerción es un mecanismo que se introduce para convertir valores de una granularidad en otra. Es imprescindible para poder refinar las características temporales de tipos relacionados mediante herencia (en ODMG no se pueden redefinir las propiedades).

Además, el modelo T_ODMG introduce la clase TimeInterval para soportar intervalos temporales de acuerdo a la noción de intervalo como una serie de instantes consecutivos entre dos instantes dados, esto es, dos valores de tipo TimeStamp, distinto a lo que implementa la clase Interval original del modelo ODMG.

En el modelo T_ODMG debemos incluso asociar un periodo de vida a una clase, con lo que podremos indicar cuando ha existido dicha clase.

3.2.3 TAU

El modelo TAU [Kako96] contiene el conjunto de todos los tipos de ODMG y los extiende añadiendo nuevos tipos de datos con semántica temporal. Los conceptos básicos introducidos son:

- tiempo de validez, tiempo de transacción
- literales, atributos, objetos y relaciones temporales
- extensión temporal
- periodo de vida
- historia de una propiedad

Además, se introduce un nuevo tipo denominado Temporal_Element que representa la unión de un conjunto finito de periodos.

El modelo no provee de tipos de objeto directamente instanciables. En lugar de ello proporciona una serie de tipos con características y semántica temporal que pueden ser incluidos en los tipos definidos por el usuario. Estos tipos de datos pueden ser estáticos (correspondientes al conjunto básico de tipos de ODMG), objetos de tipo "transaction time", objetos de tipo "valid time" y objetos de tipo "bitemporal time".

Para obtener la evolución de los valores de las propiedades el modelo utiliza los objetos históricos. Existen tres tipos de objetos históricos que se utilizan en función del tipo de objeto temporal del que guardan su evolución denominados "TT_History", "VT_History" y "BT_History", que son modelados como colecciones inmutables. Estas colecciones proveen de métodos con parámetros adecuados al tipo de objeto temporal que representan para poder obtener las instancias de los objetos correspondientes a un determinado instante o periodo.

El modelo TAU adopta el marcado temporal de cada uno de los atributos de los objetos para permitir cambios asíncronos y diferentes granularidades. Los valores se acceden a través de los métodos set_value y get_value, que son los responsables de mantener la historia de un objeto. Cada tipo de atributo temporal requiere unos parámetros diferentes. Igualmente ocurre con las relaciones, donde se definen una serie de operaciones para cada tipo de relación en función de su temporización y cardinalidad.

Un subtipo puede redefinir las propiedades y operaciones que hereda para especializarlas según el comportamiento temporal y el rango de valores apropiados de las instancias del subtipo. Para asegurar la sustitubilidad a los tipos de objeto estáticos les esta permitido tener subtipos de tipo "valid time", "transaction time" y "bitemporal time", mientras que a éstos últimos sólo les esta permitido tener como subtipos objetos de tipo "bitemporal time". En el caso de la herencia múltiple, un subtipo debe soportar todas las dimensiones temporales soportadas por su supertipo. Por ejemplo, si un subtipo T3 tiene como supertipo a un tipo "transaction time" y a otro del tipo "valid time" entonces es necesario que el tipo T3 sea declarado como "bitemporal time".

El modelo TAU extiende la definición no temporal de extensión para incluir aspectos temporales. Esta extensión puede ser especializada en "transaction time extension", "valid time extension" o "bitemporal extension". Además, especializa el concepto de periodo de vida a "transaction time lifespan " y "valid time lifespan".

3.2.4 OODAPLEX

El modelo OODAPLEX [Daya92][Wuu93] es una extensión temporal del modelo funcional DAPLEX, que extiende a éste con una serie de tipos de objeto genéricos, el tiempo y conjuntos de puntos temporales para incorporarle conceptos generales de semántica temporal. La semántica temporal más específica que requieren algunas aplicaciones se consigue mediante tipos de datos abstractos que son subtipos de estos tipos temporales genéricos.

Podemos distinguir entre objetos mutables, los cuales puede cambiar a lo largo del tiempo, y objetos inmutables. Las propiedades de estos objetos, sus relaciones con otros objetos y sus operaciones son modeladas uniformemente mediante funciones, que son aplicadas a los objetos. Las funciones pueden, de hecho, modelar el tiempo de forma unidimensional o multidimensional, simplemente añadiendo parámetros temporales a la definición de la función, por lo que podremos expresar conceptos como transaction time o valid time. Estas funciones pueden ser polimórficas, permitiendo el refinamiento de alguna de las funciones heredadas en un subtipo (pueden ser modificadas para restringir su resultado a un subtipo del resultado de la función del supertipo). Además, el modelo permite la herencia múltiple.

Cada objeto tiene asignada una función denominada periodo de vida, que devuelve el intervalo de tiempo durante el cual el objeto ha existido (asumiendo que es continuo). En

una base de datos temporal, el mismo objeto puede actuar con diferentes roles en momentos diferentes, por lo que el periodo de vida de un objeto como una instancia del tipo T puede ser diferente del periodo de vida del mismo objeto como instancia de otro tipo S.

El modelo DAPLEX también considera la semántica de sustitución, que significa que cualquier función definida sobre un supertipo puede ser aplicada a cualquier instancia de un subtipo de éste. En el modelo temporal, el resultado de la aplicación de esta función debe ser restringido temporalmente, de forma que sólo se tengan en cuenta los resultados de la función correspondientes al periodo de vida de la instancia según la clase a la que pertenezca en ese momento.

El modelo tiene la flexibilidad de soportar tanto el marcado temporal a nivel de atributo como a nivel de objeto. Incluso pueden ser mezcladas ambas aproximaciones en el mismo esquema.

Este modelo usa una aproximación para modelar el tiempo basada en el principio de separar los objetos abstractos temporales de sus correspondientes representaciones concretas. La información métrica de diferentes granularidades o precisiones puede ser asociada con puntos temporales abstractos o concretos (por ejemplo, una función "distancia" que devuelve la diferencia entre dos puntos sobre una línea temporal).

3.2.5 OSAM

OSAM*/T [Su91] es una extensión del modelo semántico, orientado a objetos OSAM* para incluir conceptos temporales e históricos. Start-time y End-time son los únicos conceptos que utiliza el modelo para guardar la historia de los objetos. Start-time es el instante en el que un nuevo objeto (o una instancia de un objeto) se vuelve activo en la base de datos (o en una clase). End-time se refiere al instante en el que un objeto (o su instancia) se vuelve inactivo en la base de datos (o clase). Otros conceptos temporales, como "transaction time" o "valid time", son capturados mediante reglas temporales. En lugar de tener un nuevo atributo para almacenar el instante en el que se graba una tupla en la base de datos, podemos definir una regla temporal que sea lanzada automáticamente antes de cada consulta a una tabla para indicar que, por ejemplo, el hecho de que el salario de Mary sea de 30.000$ se ha hecho efectivo retroactivamente el dia 12-15-87 (esto se puede hacer mediante una cláusula condicional que devuelva un resultado concreto en función del intervalo de la consulta que se esta realizando). Esta aproximación permite definir gran cantidad de requerimientos temporales y evita el excesivo requerimiento de espacio de almacenamiento y operaciones de entrada/salida en el sistema de gestión de base de datos que requiere la primera aproximación.

La técnica de marcado temporal de los objetos seguida en este modelo es un compromiso entre las aproximaciones de marcado temporal de atributos y marcado de la tupla completa. Cada una de las instancias de los objetos se marca con un intervalo temporal, <

Start-time, End-time>. Cuando se produce un evento que afecta a una instancia de un objeto, su valor End-time se actualiza al instante en el que se produce el evento, la instancia se vuelve inactiva y pasa al área histórica. Cuando se crea una nueva instancia de un objeto debido a un evento, todos los atributos que no son afectados por este evento se marcan con el símbolo "#" en lugar de guardar valores repetidos de la instancia histórica. La unidad temporal escogida para una aplicación puede, además, ser diferente que la escogida para otra aplicación, dependiendo de la granularidad necesitada.

Una instancia se identifica unívocamente mediante un identificador (IID), el cual se forma mediante la concatenación del identificador de la clase y el identificador del objeto. Si un objeto es instancia de más de una clase a lo largo del tiempo, cada instancia del mismo en cada clase tendrá su propia historia.

El modelo provee de cinco tipos de asociación predefinidas entre tipos (agregación, generalización, interacción, composición y producto cruzado) que pueden usar tanto el diseñador como los usuarios de la base de datos para especificar diferentes relaciones semánticas o asociaciones entre las clases y sus instancias. Las propias relaciones son tratadas como instancias de una clase de objetos. Una de las ventajas del marcado temporal a nivel de objeto es que la historia de las asociaciones entre las instancias de los objetos puede ser derivada a partir de la historia de las propias instancias.

3.2.6 TEMPOS

TEMPOS [Duma04] es un conjunto de modelos y lenguajes para el soporte y manipulación de datos temporales. Se ofrecen dos formas diferentes para acceder a los datos temporales: un lenguaje de consulta y un navegador visual. El objetivo del proyecto TEMPOS (Temporal Extension Models for Persistent Object Servers) es contribuir hacia una forma consensuada sobre como manejar la temporalidad en modelos orientados a objetos, definiendo un marco dentro de las bases de datos para los objetos temporales.

El modelo de datos TEMPOS se basa en un conjunto de tipos de datos cuyo comportamiento se describe a través de tipos de interfaces. La distinción entre interfaces (descripción abstracta) y clases (implementación concreta) se justifica en que es necesario la separación entre la semántica de las operaciones sobre los tipos de datos y su implementación sobre una representación concreta. Además, TEMPOS se estructura en tres niveles, cada uno de ellos conteniendo al anterior:

- El primer nivel se compone de un conjunto de tipos de datos que modelan valores de tiempo(instantes, intervalos y conjunto de instantes), expresados en múltiples granularidades (unidades de tiempo). Con este nivel tenemos posibilidad de representar aplicaciones que conlleven asociaciones temporales tales como eventos con fecha (cumpleaños, citas, etc.).
- El segundo nivel introduce el concepto de historia y sus operadores asociados. Estos operadores se utilizan por los programas de aplicación para filtrar,

transformar y agregar datos históricos. En nivel es apropiado para aplicaciones que necesitan manipular asociaciones temporales complejas pero no requieren ningún soporte específico para realizar actualizaciones teniendo en cuenta el tiempo de transacción o el tiempo de validez.

- El tercer nivel extiende el concepto de clase, atributo y relación tal y como están definidas en ODMG standard, y maneja los conceptos de clase temporal, atributo temporal y relación temporal. Este nivel puede manejar tiempo de transacción o el tiempo de validez.

TEMPOS se basa en un modelos de tiempo discreto, lineal y acotado en el que el eje tiempo se estructura de forma multigranular a través de unidades de tiempo. Una unidad de tiempo modela una partición de la línea de tiempo en intervalos disjuntos e intervalos contiguos.

Se definen cuatro tipos temporales básicos: instante, duración, intervalo y conjunto de instantes. A partir de estos tipos básicos el modelo histórico propuesto proporciona la capacidad para capturar la evolución de los datos a lo largo del tiempo.

Una *historia* se define de forma abstracta con una función que va de un conjunto de instantes finitos a un conjunto de valores de un determinado tipo. El conjunto de historias de un tipo T se modela con un tipo de datos abstracto *History*(T). Las historias se pueden representar de varias formas, principalmente a través de colecciones de valores con marca de tiempo, llamados *cronos*.

Como en ODMG, una propiedad se define como un atributo o camino transversal asociado a una clase. En TEMPOS, una propiedad puede ser *temporal* lo que implica que los sucesivos valores se van almacenando, o puede ser *fugaz* en cuyo caso solo se almacenan los valores más recientes. Cuando una propiedad es temporal, la granularidad con la que evoluciona se determina por las características específicas de la propiedad, llamadas su *unidad de observación*.

TEMPOS distingue entre tiempo de transacción y tiempo de validez. Mezclando estos conceptos es posible obtener asociaciones bitemporales.

3.2.7 TRIPOD

Tripod [Grif04] es un modelo orientado a objetos espacio-temporal basado en un mecanismo de especialización, llamado una historia, para mantener el conocimiento de las entidades que cambian a lo largo del tiempo. El proyecto Tripod esta desarrollando un sistema de base de datos completamente espacio-temporal. El principal objetivo es la extensión del estandar ODMG para bases de datos orientadas a objetos con herramientas para el manejo de vectores datos espaciales y para la descripción de los estados pasados de datos tanto espaciales como no espaciales. Los principios claves del proyecto Tripod son la ortogonalidad y la sinergia. Por ortogonalidad se entiende que las diferentes

extensiones del estándar ODMG deben ser coherentes por separado, por lo que, por ejemplo, el sistema Tripod debe ser efectivo tanto como una base de datos histórica en donde no se almacenan datos espaciales como en bases o como una base de datos espacial en donde no se hace usos de sus características para guardar datos históricos. Por sinergia se entiende que el sistema debe permitir el uso combinado de las capacidades espaciales y temporales de una manera correcta y complementaria. Una base de datos Tripod en la que no se hace uso de las características espaciales ni temporales es una base de datos ODMG, funcionalmente comparables a las que existen en el mercado.

En el núcleo se encuentra el modelo de datos orientado a objetos ODMG, que el modelo Tripod extiende con dos nuevas categorías de tipos primitivos: tipos espaciales y tipos de marcado temporal (timestamp types). Los tipos espaciales son aquellos del álgebra ROSE (Robust Spatial Extensions), esto es, los tipos vectoriales Punto, Línea y Región. Los tipos de datos de marcado temporal son versiones unidimensionales de los tipo de datos del álgebra ROSE Puntos y Líneas, referenciados como Instantes e IntervalosTemporales, respectivamente. La estrecha relación entre los tipos espaciales y de marcado temporal incrementa la consistencia en la representación de diferentes tipos de datos. Estados pasados de objetos que se refieren a algún tipo ODMG, incluyendo los espaciales y de marcado temporal, pueden ser almacenados usando un mecanismo especializado denominado una historia. En esencia, una historia es una colección de pares de valores temporalmente marcados, donde la marca temporal es del tipo marca temporal y el valor es de alguno de los tipos del modelo de datos resultante de extender ODMG. El modelo Tripod extiende los tipos de datos literales de ODMG con seis nuevos tipos literales estructurados para representar información espacial. Estos tipos de datos espaciales (SDTs) se basan en el álgebra ROSE.

Tripod extiende también el conjunto de tipos primitivos de ODMG con dos tipos de datos de marcado temporal, denominados Instante e IntervaloTemporal. El dominio de interpretación subyacente es una estructura a la que nos referimos como un área temporal porque es definida como una especialización unidimensional de áreas bidimensionales (espaciales). Aproximadamente, un área temporal es un conjunto finito de enteros (mientras que un área espacial es u conjunto finito de parcelas.

En un área temporal, podemos pensar en un punto en el tiempo como un entero. Entonces, un valor de tipo Instante es una colección de puntos temporales y un valor de tipo IntervaloTemporal es una colección de pares de valores de puntos temporales, donde el primer valor es el inicio y el segundo el fin de un intervalo temporal contiguo. Una marca temporal es un valor de tipo Instante o de tipo IntervaloTemporal. Tripod adopta el calendario estándar Gregoriano, permitiendo que las marcas temporales se declaren en alguna de las siguientes colecciones de granularidades: Año, Mes, Día, Hora, Minuto y Segundo.

El dominio discreto que subyace al área temporal esta enmarcado entre dos valores enteros que corresponden a los valores de tipo marca temporal más antiguo y más

reciente que se pueden representar. Estos valores tienen un nombre especial, "comienzo" y "eternidad", que pueden ser referenciados como el resto de literales de tipo marca temporal de Tripod.

La colección completa de tipos literales de Tripod puede ser vista como un superconjunto de todos aquellos especificados en el estándar ODMG 3.0, con los tipos de datos de marcado temporal de Tripod suplantando los tipos Timestamp, Date y Time de ODMG.

El mecanismo de historia de Tripod provee de la funcionalidad necesaria para dar soporte al almacenamiento, manejo y consulta de las entidades que cambian a lo largo del tiempo. Una historia modela los cambios que una entidad (o sus atributos o las relaciones en las que participa) sufre como resultado de aplicar una nueva asignación sobre ella. En el modelo Tripod, una petición para el mantenimiento de una historia puede ser realizada para cualquier constructor al que se le pueda asignar un valor, esto es, una historia es una historia de los cambios en un valor y almacena los episodios de cambios identificándolos con una marca temporal. Cada uno de estos valores se denomina captura. Como consecuencia de las posibles asignaciones de valores que están bien definidos en el modelo Tripod, una historia puede reservarse para identificadores de objetos, valores de atributos e instancias de las relaciones.

La extensión histórica de Tripod provee de la capacidad para almacenar la historia de cambios que una entidad ha sufrido a lo largo del tiempo, extendiendo modelo ODMG con la habilidad de poder seguir los cambios causados por operaciones de asignación en cualquiera de los constructores de ODMG que sea asignable y que sea declarado como histórico.

Para cada concepto de ODMG que es asignable, el modelo Tripod provee de un concepto histórico correspondiente, de la siguiente forma: tipos de objetos históricos atómicos, atributos históricos, relaciones históricas y colecciones históricas.

Además de poder especificar que un concepto de la base de datos debe ser mantenido históricamente, el diseñador de la aplicación puede especificar también ciertas propiedades de la historia, denominadas:

- Granularidad: Si una historia tiene granularidad C, entonces no puede haber grabada más de una captura por cada un de los granos de C.
- Tipo marca temporal: Cada cambio es asociado con una marca de tipo Instante o IntervaloTemporal.

En el modelo Tripod, cualquier tipo de objeto (incluidos los definidos por el usuario) pueden ser declarados históricos. Esta declaración implica la creación de un atributo extra, cuyo valor es una historia y al que se puede hacer referencia en consultas y programas de aplicación como "tiempo de vida", de forma automática por el sistema y que éste debe encargase de mantener para cada una de las instancias del tipo en cuestión. El atributo "tiempo de vida" almacena el estado del objeto para un valor de tipo Instante

o IntervaloTemporal concreto. La captura en cada estado del "tiempo de vida" indica si el objeto se encontraba activo o inactivo para una marca temporal especificada. Esta capacidad permite activar y desactivar objetos en una determinada base de datos, en lugar de tener que crearlos y eliminarlos. Por defecto, todos los objetos son creados con el intervalo [Ahora, Hasta que cambie).

Cualquier propiedad (atributo o relación) de un tipo de objeto atómico puede ser declarada como histórica. Por ejemplo, el siguiente código declara un atributo histórico y una relación histórica para un tipo de objeto atómico Edificio.

```
Historical (timeIntervals; DAY) attribute regions geographical extent;
Historical (timeIntervals; DAY) relationship Burglary undergoes
inverse Burglary::is_of;
```

Una instancia de una colección de tipos de objetos es un objeto con nombre cuyos elementos que lo componen son del mismo tipo. Este tipo puede ser un tipo de dato atómico, otra colección o un literal. Un tipo de objeto colección histórica utiliza el mecanismo historia para permitir que se pueda almacenar la historia de sus cambios.

La herencia implica importantes consideraciones a la hora de definir la especialización de un tipo que tiene que ver con declaraciones de tipos históricos. Dado que la intención de la noción de herencia es permitir la especialización de un tipo en un subtipo, proveyendo de una información más específica, entonces, en el modelo Tripod un subtipo sólo puede tener una especialización histórica más precisa que su supertipo. Por lo tanto, un tipo histórico no puede ser especializado en un tipo de dato no histórico, pero si al contrario.

En una base de datos ODMG, una actualización de una relación provoca una actualización recíproca de la relación inversa. En Tripod, este comportamiento se extiende para conseguir relaciones históricas. A pesar de ello, esta capacidad es substancialmente mas compleja que en el caso de las relaciones no históricas y requiere del uso y coordinación de operaciones sobre las historias involucradas.

En Tripod, las relaciones sólo pueden existir entre tipos de objetos y no entre tipos de objetos y tipos literales. Además, Tripod siempre mantiene relaciones inversas de todas las relaciones de los objetos.

En la Tabla 3 se resumen las principales características de los modelos.

Nombre	Modelo Datos	Nivel Temp	Densidad de Tiempo	Tipos Temporales	VT	TT	Extiende ODMG	Otros tipos Tem
T_CHIMERA	Objeto	1, 2, 3	Discreto	Instante, intervalo	Sí	Sí	Sí	
T-ODMG	Objeto	1, 2, 3	Discreto	Duración, Instante, intervalo	Sí	Sí	Sí	
TAU	Objeto	1, 2, 3	Discreto	Duración, Instante, intervalo	Sí	Sí	Sí	
OODAPLEX	Objeto	1, 2	Definido por el usuario	Instante	Sí	Sí	No	
OSAM	Objeto	1	Discreto	Instante, intervalo	Sí	No	No	
TEMPOS	Objeto	1, 2, 3	Discreto	Duración, instante, intervalo	Sí	Sí	Sí	TSequence (set of instants), chronicles
TRIPOD	Objeto	1, 2, 3	Discreto	Instante, intervalo			Sí	Points, Lines and Regions

Nivel temporal: 1. Marcado a nivel de Objeto ; 2. Marcado a novel de atributo; 3. Marcado a nivel de relación.

Tabla 3. Resumen de modelos temporales

3.3 Modelo de tiempo

Teniendo en cuenta los modelos anteriormente expuestos, a continuación, definiremos nuestro modelo.

Existen diferentes formas de representar las entidades temporales. Nuestra propuesta se basa, fundamentalmente, en un modelo de tiempo lineal y discreto en el cual el tiempo se estructura con múltiples granuralidades, que forman un subconjunto de los números naturales. El eje de tiempo se divide en un conjunto finito de pequeños segmentos llamados gránulos. El gránulo más pequeño es un chronon y su tamaño depende de las necesidades de diseño e implementación. En nuestro modelo se definen las siguientes entidades temporales:

- *Instante (instant)*: la continuidad del tiempo puede modelarse como una línea recta formada por un conjunto infinito de puntos Ti denominados instantes. Es una colección totalmente ordenada mediante la relación '≤'.
- Una *marca de tiempo (timestamp)* es un valor de tiempo asociado con un objeto, por ejemplo un atributo o una tupla.
- *Unidad temporal (temporal unit)* : es una partición del tiempo en un conjunto de intervalos de tiempo disjuntos donde cada intervalo es de granularidad atómica. Por ejemplo: día, mes, año.

- *Intervalo (period):* una sección de la línea de tiempo, determinada por la duración entre dos instantes.
- *Evento (event)*: es un hecho que se produce en la realidad y que provoca un cambio en el estado de, al menos, una fuente de datos. Un evento tiene lugar en un punto particular de la línea de tiempo Ti y, por tanto, no tiene duración.
- *Tiempo de vida (lifespan)*: es el tiempo que un objeto está definido. El tiempo de validez de vida de un objeto corresponde con el periodo de tiempo en el cual el objeto existe en la realidad modelada. El tiempo de transacción de vida un objeto corresponde con el tiempo en el cual el objeto está grabado en el sistema operacional.
- *Tiempo de transacción (Transaction Time, TT)*: es el intervalo de tiempo durante el cual, un hecho esta disponible en las fuentes de datos.
- *Tiempo de validez (Valid Time, VT)*: corresponde con el intervalo de tiempo en el que un hecho es cierto en la realidad que se modela. Un determinado evento genera un intervalo de validez y un intervalo de transacción. Generalmente VT es proporcionado por el usuario.
- *Objeto temporal*: se trata de objetos (clases, interfaces, …) que tenga una semántica temporal diferenciada de los objetos no temporales.
- *Granularidad (granularity)*: nivel de detalle con que son marcados los datos. Por ejemplo, si tenemos una granularidad día, implica que tenemos una snapshot por registro y día.

Con estos conceptos, es posible representar perfectamente todas las propiedades temporales de las fuentes de datos (metadatos temporales), todas las características deseables en la extracción, y todos los elementos temporales en los esquemas de las fuentes y del almacén.

3.4 El standard ODMG

Como Modelo Canónico de Datos, hemos optado por el modelo de objetos de ODMG. Este modelo será extendido con conceptos temporales, de manera que se puedan representar dichas características en los esquemas de las fuentes de datos y del almacén de datos, y en los metadatos de las fuentes y métodos de extracción utilizados en éstas. A continuación, presentamos un resumen de las principales características del estandar ODMG y, en particular de su modelo de objetos.

3.4.1 Introducción

Un concepto clave de un sistema de bases de datos es el modelo de datos en el que está basado. Las bases de datos orientadas a objetos, a diferencia de las bases de datos relacionales, no han estado basadas en un modelo de objetos estándar. Ante esta carencia, ODMG (*Object Data Management Group*), un consorcio formado por las principales

compañías relacionadas con el desarrollo de productos de bases de datos orientadas a objetos y objeto-relacionales, comenzó a desarrollar en 1991 un estándar para bases de datos de objetos, de forma que los programadores pudiesen construir aplicaciones de bases de datos que utilizasen objetos siguiendo una definición común. Esta definición común permite a los programadores escribir aplicaciones portables, por lo que los lenguajes de definición y consulta, y las ligaduras a los lenguajes de programación incluidos en el estándar proporcionan un marco de trabajo para la definición de este tipo de aplicaciones.

La primera especificación del estándar ODMG se publicó en 1993 bajo el nombre de ODMG-93. Posteriormente, en 1997 apareció una versión revisada, ODMG 2.0, que incluía soluciones para algunas carencias de la versión anterior, y desde 2000 está disponible ODMG 3.0. Las especificaciones de ODMG están formadas por un modelo de objetos, un lenguaje de definición de objetos (ODL), un lenguaje de consulta de objetos (OQL) y distintas ligaduras para distintos lenguajes de programación, como C++, Java y Smalltalk.

3.4.2 Evolución de las especificaciones de ODMG

Desde su aparición en 1993, las especificaciones del estándar ODMG han sufrido distintas modificaciones añadiendo mejoras al estándar y nuevas características para dar cobertura a los problemas de la industria de la orientación a objetos. Actualmente, en ODMG están representadas la mayoría de las compañías relacionadas con productos de bases de datos orientadas a objetos. En esta sección vamos a ofrecer una visión general de la evolución que han sufrido las especificaciones de ODMG desde su versión inicial hasta la versión actual; asimismo, enumeraremos los principales miembros de ODMG.

3.4.2.1 Evolución del estándar ODMG

Desde que un grupo de empresas decidieran comenzar a definir las especificaciones que sirviesen como base para la creación de un estándar para bases de datos orientadas a objetos en el verano de 1991, pasaron casi tres años hasta que se editó la primera versión del estándar ODMG, ODMG-93 [Cate94a]. Actualmente, y tras haber sufrido algunas críticas [Kim94, Alag97, Alag99], estas especificaciones están en su tercera versión, ODMG 3.0 [Cate00], siendo actualmente el estándar de hecho de la industria. No obstante, el estándar ODMG aún no ha sido reconocido como tal por ningún comité de estandarización, pero tal y como se indica en [Cate95], son tantos los intereses enfrentados que es necesario contar un estándar de hecho en el que se base la definición de un estándar real.

Las especificaciones de ODMG constan de varios componentes que serán tratados a lo largo de este capítulo. Dichos componentes son un modelo de objetos, un lenguaje de

definición de objetos, un lenguaje de consulta de objetos y ligaduras a distintos lenguajes de programación.

El modelo de objetos de ODMG está basado en el modelo de objetos de OMG (*Object Management Group*)[2] incluyendo características de bases de datos, como son las relaciones (*relationships*) y extensiones (*extents*). El lenguaje de definición de objetos ODL (*Object Definition Language*) es un superconjunto del lenguaje de definición de interfaces IDL (*Interface Definition Language*) de OMG, y se utiliza para definir esquemas de bases de datos en términos de tipos de objetos, atributos, relaciones y operaciones, sin hacer referencia a ningún lenguaje de programación. El lenguaje de consulta de objetos OQL (*Object Query Language*) es un lenguaje de consulta basado en la sintaxis del SELECT de SQL con extensiones para el soporte de objetos, incluyendo identidad de objetos, objetos complejos, expresiones *path*, llamada a operaciones y herencia. No obstante, ODMG mantiene relaciones con el comité que trabaja en SQL3 para que OQL sea compatible con la parte de consulta de SQL3. Por último, las ligaduras a lenguajes son extensiones de diferentes estándares de lenguajes de programación para permitir el almacenamiento de objetos persistentes, incluyendo además soporte para OQL, navegación y transacciones.

Cada uno de estos componentes ha sufrido diferentes modificaciones a lo largo de estas versiones, de forma que el estándar se corresponda con las necesidades de los desarrolladores y fabricantes de software que utilizan aplicaciones que manejan objetos persistentes.

En ODMG-93 [Cate94a], se presentaba un estándar cuyos componentes principales eran un modelo de objetos, un lenguaje de definición de objetos ODL, un lenguaje de consulta de objetos OQL, y las ligaduras para C++ y Smalltalk. Pese a que se distinguía entre los conceptos de tipo y de clase, sólo existían declaraciones `interface`, mediante las cuales se especificaban las clases que formaban los esquemas. Estas definiciones `interface` podían incluir un nombre para la extensión (`extent`). La primera versión del estándar ODMG no incluía en sus especificaciones información sobre los metadatos e incorporaba una serie de tareas pendientes para la siguiente versión, entre las que cabe mencionar en esta tesis la incorporación del soporte necesario para la definición de *vistas*, haciendo referencia a la definición de clases derivadas.

En ODMG 2.0 [Cate97] ya aparece de forma clara la división entre interfaz y clase, de forma que los esquemas pueden incluir definiciones `interface` y definiciones `class`. Las interfaces se corresponden con especificaciones de tipo que definen comportamiento abstracto, mientras que las clases definen estado y comportamiento abstracto. Esta división da lugar a dos tipos de herencia, de forma que la relación de herencia que puede existir entre dos interfaces, o bien entre una clase y una interfaz es una relación `ISA`; la relación de herencia que puede existir entre dos clases es una relación `EXTENDS`. En esta

[2] OMG es un consorcio de empresas fundado en 1989 para promover el uso de la orientación a objetos y la definición de una arquitectura para el desarrollo de programas basados en objetos distribuidos.

segunda versión del estándar ODMG se describen los metadatos que definen las metaclases del esquema del repositorio de ODMG. Además, se introduce el concepto de *consultas con nombre*, como una característica que permite la definición de *vistas*, haciendo referencia a las vistas relacionales. En cambio, dichas consultas con nombre sólo permiten almacenar la definición de una consulta, pero no se pueden utilizar como mecanismo para la definición de clases derivadas, tal y como veremos al final de este capítulo. También aparecen por primera vez en las especificaciones de ODMG las interfaces para definición de bases de datos y de esquemas como son las interfaces `Database` y `Module`, respectivamente. En las especificaciones de ODMG 2.0 se incluye un lenguaje para el intercambio de objetos entre bases de datos, OIF (*Object Interchange Format*), que permite volcar y cargar el estado de una base de datos orientada a objetos en o desde un conjunto de archivos. Por último, y como otra incorporación notable se incluye la ligadura para Java, y se extiende la ligadura de C++ para la inclusión de metadatos.

La versión 3.0 del estándar ODMG [Cate00] no incluye grandes modificaciones sobre la versión 2.0, aunque hay ciertos aspectos diferentes, que comentamos a continuación. En la versión 3.0 se corrigen algunos errores que producían cierta confusión en el uso de interfaces y clases. Asimismo, se realiza una ligera actualización de los metadatos, y se utilizan de forma explícita los módulos, ya que hasta la versión 3.0 los esquemas aparecían como una lista de clases e interfaces pero sin pertenecer a ningún tipo de módulo o esquema. En esta versión también se incluye la gramática del lenguaje de intercambio de objetos (OIF) que, pese a que el lenguaje se propuso en la versión 2.0, es en la versión 3.0 donde se presenta su gramática. Por último, se han realizado modificaciones para la ligadura de Java con el fin de adaptarla a Java 2, y estas especificaciones están siendo evaluadas por la Java Community Process para formar la base de la *Java Data Objects Specification.*

3.4.2.2 Miembros de ODMG

Las compañías que forman parte actualmente de ODMG representan a la mayor parte de la industria de las bases de datos de objetos. Desde su nacimiento en 1991, el consorcio de ODMG ha estado creciendo constantemente y entre sus miembros están vendedores de bases de datos, vendedores de herramientas, consultores y empresas que utilizan productos que siguen las especificaciones de ODMG. No obstante, el grado de pertenencia al consorcio de ODMG varía en función del tiempo que dedique un delegado de la empresa a actividades relacionadas con ODMG, y en función de la utilización del estándar ODMG por parte de la empresa. Así, podemos encontrar miembros con derecho a voto (*voting members*) y revisores. Además, varias universidades e institutos están involucrados en el desarrollo de este estándar.

Los miembros con derecho a voto son empresas o instituciones que utilizan ODMG y en las que un delegado de la empresa dedica más del 20% de su tiempo a actividades relacionadas con ODMG. Actualmente, entre los miembros con derecho a voto podemos

encontrar a Ardent Software, Ericsson, eXcelon Corporation, Object Design, Objectivity, POET Software Corporation, Sun Microsystems, Versant Corporation y Winward Solutions. Los revisores son entidades que dedican un 10% del tiempo de un delegado a actividades relacionadas con ODMG, pero no tienen derecho a voto. Actualmente, podemos encontrar como revisores de ODMG a Advanced Languages Technologies, Baan, CERN, Computer Associates, Gemstone Systems, Hitachi, Micro Data Base Systems (mdbs), Lawson Software, NEC Corporation, Telenor R&D. En lo que respecta a miembros académicos se encuentran representantes de distintos institutos y universidades, así como investigadores tan destacados en bases de datos orientadas a objetos como Stan Zdonik, Klaus R. Dittrich y Suad Alagic.

Al final de este capítulo, se ofrece una relación de distintos productos que cumplen en mayor o menor medida las especificaciones de ODMG, indicando qué parte de las especificaciones de ODMG son implementadas por dichos productos.

3.4.3 El modelo de objetos de ODMG

El modelo de objetos especifica la semántica que se puede definir en una base de datos orientada a objetos. Entre otras cosas, esta semántica determina las características de los objetos, cómo se relacionan entre sí y cómo se pueden denominar e identificar.

El modelo de objetos de ODMG es el núcleo del estándar ODMG y está basado en el modelo de objetos común (*Common Object Model*) de OMG. El modelo de objetos común de OMG se creó como un denominador común para gestores de peticiones de objetos (*Object Request Brokers*), sistemas de almacenamiento de objetos, lenguajes de programación de objetos y otras aplicaciones. El modelo de objetos de ODMG añade componentes al modelo de objetos común de OMG como son las relaciones y las transacciones.

Las primitivas básicas de modelado en el estándar ODMG son el *objeto* y el *literal*. La diferencia entre ambas primitivas está en el concepto de identidad de objeto, de forma que cada objeto tiene un identificador único, mientras que los literales no tienen identificador. El hecho de que los objetos tengan identificador hace que sean denominados *mutables*, mientras que el que los literales no tengan identificador y estén caracterizados únicamente por su valor hace que sean *inmutables*. Los objetos pueden modificar su valor sin modificar su identidad mientras que la modificación de un literal supone la creación de un nuevo literal, de ahí lo de mutabilidad e inmutabilidad.

Los objetos y literales se pueden categorizar por sus *tipos*. Todos los elementos de un tipo determinado tienen un rango de estados común (el mismo conjunto de propiedades) y un comportamiento común (el mismo conjunto de operaciones). A un objeto también se le suele conocer como *instancia* de un tipo. Así pues, tendremos *tipos objeto*, cuyas instancias son objetos, y *tipos literal*, cuyas instancias son literales.

Los objetos tienen una serie de propiedades que los caracterizan. Los valores que toman estas propiedades para un objeto definen su *estado*. Estas propiedades pueden ser *atributos*

del propio objeto (p.e. nombre, idEmpleado) o *relaciones* con otros objetos (p.e. relación entre profesores y alumnos). En ODMG sólo se permiten relaciones binarias, que pueden ser 1:1, 1:M o M:N, dependiendo del número de instancias de cada tipo que puedan participar en la relación. Por ejemplo, podemos definir una relación 1:1 *director* entre instancias del tipo *Departamento* e instancias del tipo *Profesor*, una relación 1:M *miembros* entre instancias del tipo *Departamento* e instancias del tipo *Profesor* o una relación M:N *matriculados* entre instancias del tipo *Asignatura* e instancias del tipo *Alumno*. El *comportamiento* de un objeto está determinado por el conjunto de operaciones que puede realizar o que pueden ser realizadas sobre él. Dichas operaciones pueden producir *excepciones*, que se corresponden con condiciones de error que pueden producirse.

La especificación de un tipo consiste en una descripción abstracta, independiente de su implementación, de las operaciones, propiedades y excepciones visibles a los usuarios de un tipo. La especificación de un tipo se puede realizar de varias formas. Una definición `interface` es una especificación que sólo define el comportamiento abstracto de un tipo objeto. Una definición `literal` sólo define el estado abstracto de un tipo literal. Una definición `class` es una especificación que define el comportamiento abstracto y el estado abstracto de un tipo objeto. Los objetos no pueden crearse directamente a partir de un tipo `interface` sino que se tienen que crear a partir de un tipo `class`. En el modelo de objetos de ODMG las operaciones se definen siempre en un tipo y sólo en uno, pero un nombre sólo tiene que ser único en la definición del tipo.

Por ejemplo, una interfaz `Worker` definiría sólo el comportamiento común de objetos `Worker`. Una clase `People` definiría el estado y el comportamiento de objetos `People`. Por último, la estructura `Complex` definiría sólo el estado abstracto de literales correspondientes a números complejos.

```
interface Worker {. . .};
class People {. . .};
struct Complex { float re; float im; };
```

El modelo de objetos de ODMG introduce el concepto de interfaz para poder expresar el comportamiento común entre dos tipos que no tienen por que estar relacionados mediante herencia. Por ejemplo, podemos considerar una base de datos en la que dispongamos de dos clases para empleados, una para los empleados de plantilla y otra para los empleados temporales, de forma que ambas clases compartan ciertas características, pero sin que exista una relación de subclase entre ellas. Este es el origen de las interfaces, que permiten la definición de un tipo con dichas características comunes a partir del cual se pueden definir otras clases e interfaces.

Debido a las distintas especificaciones de tipos objeto que se pueden llevar a cabo en ODMG (interfaz y clase), el estándar ODMG distingue entre dos tipos de herencia, denominadas *herencia de comportamiento* (`ISA`) y *herencia de estado y de comportamiento* (`EXTENDS`). La herencia de comportamiento (`ISA`) permite establecer una relación entre tipos que permite crear tipos más especializados a partir de tipos existentes. Esta relación de

herencia de comportamiento entre tipos objeto es una relación de herencia múltiple que sólo define relaciones de herencia entre el comportamiento de tipos objeto. La relación EXTENDS define relaciones de herencia de estado y comportamiento entre tipos objeto, pero es una relación de herencia simple entre dos clases, donde la clase subordinada hereda todo el estado y todo el comportamiento de la clase a la que extiende. La relación que puede existir entre dos interfaces o bien entre una clase y una interfaz a partir de la que se define la clase es una relación ISA, mientras que la relación que puede existir entre dos clases es la relación EXTENDS. La relación ISA se especifica añadiendo dos puntos al nombre de la interfaz o clase seguido de la lista de interfaces que actúan como tipo base. En cambio la relación EXTENDS se representa añadiendo la palabra reservada `extends` al nombre de la clase que actúa como subclase y escribiendo a continuación el nombre de la clase a la que extiende.

Por ejemplo, puede existir una relación ISA entre una clase `Temporary` que esté basada en la interfaz `Worker`. Asimismo, puede existir una relación EXTENDS entre una clase `Client` y una clase `People`:

```
class Temporary: Worker {. . .};
class Client extends People {. . .};
```

La extensión de un tipo es el conjunto de instancias del tipo en una base de datos concreta. Si un objeto es instancia de un tipo A, entonces es miembro de la extensión de A. Si el tipo de A es un subtipo del tipo B, entonces la extensión de A es un subconjunto de la extensión de B. En ODMG, sólo las clases pueden tener extensión, y la extensión de una clase representa el conjunto de objetos persistentes que pertenecen a dicha clase.

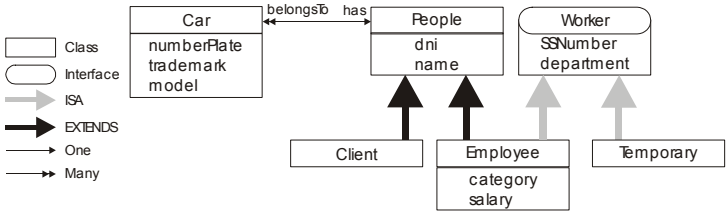

Figura 23. Esquema conceptual

La Figura 3.1 ilustra un esquema ODMG de una base de datos de personas. Las personas pueden ser clientes o empleados, y ambos poseen vehículos. Los clientes tienen las mismas características que las personas, pero sobre los empleados se guardan algunas características más. También se guarda información sobre los trabajadores eventuales. Los eventuales y los empleados de plantilla tienen propiedades y comportamiento en común, pero diferentes de los rasgos comunes entre empleados y clientes. Para simplificar, en la figura no se incluyen operaciones.

3.4.4 Principales metadatos de ODMG

Los metadatos en una base de datos, entre otra información, representan información descriptiva acerca de los objetos que forman los distintos esquemas de la base de datos. En el estándar ODMG dicha información se guarda en el *repositorio de esquemas*, cuya estructura (metaesquema) también es un esquema orientado a objetos. En esta sección describiremos los metadatos más importantes que definen la estructura interna del repositorio de esquemas. Las definiciones correspondientes a los metadatos están agrupadas en un módulo que define un ámbito de nombres para los elementos del modelo. Dicho módulo se denomina `ODLMetaObjects`.

En ODMG, todos los objetos del repositorio son subclases de tres interfaces: `MetaObject`, `Specifier` y `Operand`. Los metadatos de `Specifier` y `Operand` representan conceptos que no afectan al tema tratado en esta tesis, como son, respectivamente, la asignación de nombres a tipos en ciertos contextos y la definición de valores constantes. Por tanto, sólo prestaremos atención a `MetaObject`. La Figura 3.2 ilustra la parte del metaesquema del estándar ODMG 3.0 que representa a los metadatos que vamos a considerar en esta tesis. En dicha figura se puede observar que los metadatos forman parte de un esquema compuesto por metaclases. Las instancias de dichas metaclases son los distintos tipos, clases, relaciones y demás que se utilizan en la definición de esquemas.

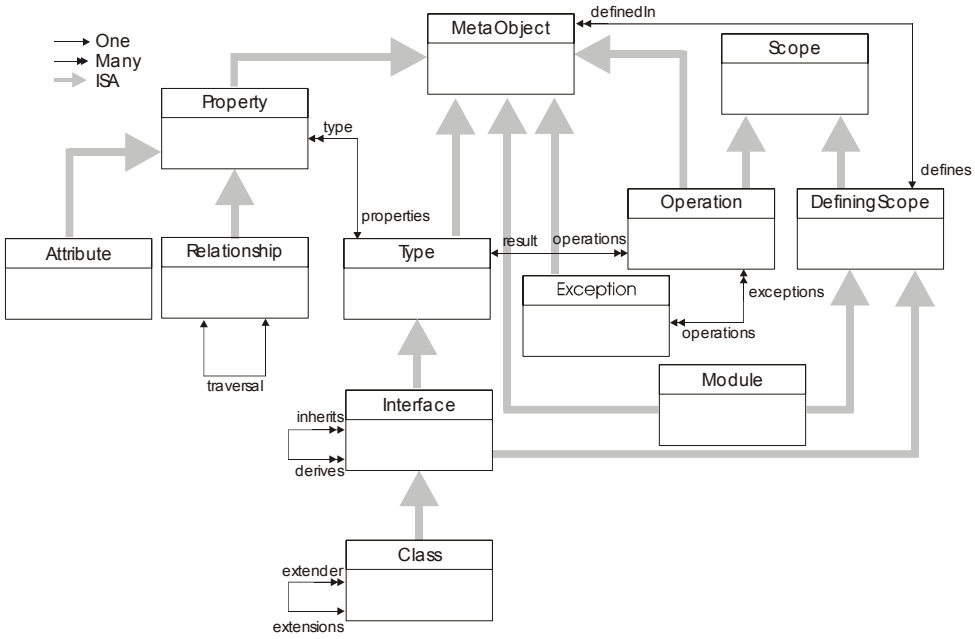

Figura 24. Fragmento del metaesquema de ODMG

A continuación se describen brevemente los principales metadatos de la Figura 3.2.

- `MetaObject`. Los metaobjetos representan los elementos del esquema guardados en el repositorio. Todos los metaobjetos tienen un nombre y participan en una relación `definedIn` con otros metaobjetos que establecen sus ámbitos de definición (p.e. la definición de una clase en un módulo o la definición de un atributo en una clase).
- `Scope`. Los objetos de los esquemas están definidos en un ámbito (*scope*). Los ámbitos definen una jerarquía de nombres para los metaobjetos del repositorio. Una instancia de `Scope` contiene una lista de las instancias de `MetaObject` (metaobjetos) que se encuentran definidas en su ámbito. Así pues, un módulo, que en ODMG define un ámbito de definición de clases, interfaces y excepciones, contiene una lista de las clases, interfaces y excepciones que se han definido en dicho módulo.
- `Module`. En ODMG, los módulos (*modules*) y el propio esquema del repositorio, que es un tipo de módulo, son ámbitos de definición (*DefiningScope*). Estos metaobjetos incluyen operaciones para la incorporación de los módulos, clases e interfaces que forman los módulos. Los módulos se utilizan para agrupar ciertas instancias de `MetaObject`, como interfaces, clases y demás. Las instancias de `Module` ocupan la parte superior en la jerarquía de nombres.
- `Type`, `Interface` y `Class`. Las instancias de `Type` se utilizan para representar información sobre los tipos de datos. Las instancias de `Interface` definen el comportamiento abstracto de los objetos de la aplicación. Las interfaces están relacionadas en un grafo de herencia múltiple con otras interfaces mediante las relaciones `inherits` y `derives`. `Class` es un subtipo de `Interface`. Sus propiedades definen el estado de los objetos guardados en la base de datos. Las clases forman una jerarquía de herencia simple por medio de las relaciones `extender` y `extensions`, de forma que el estado y el comportamiento se heredan de las clases que son extendidas.
- `Property`, `Attribute` y `Relationship`. `Property` es una clase abstracta a partir de la que se definen `Attribute` y `Relationship`, que determinan el estado abstracto de un objeto de la aplicación. Las propiedades se definen en el ámbito de una estructura o de una clase. `Attribute` representa a las propiedades que mantienen el estado abstracto. `Relationship` modela la asociación entre objetos persistentes.
- `Operation` y `Exception`. Las instancias de `Operation` modelan el comportamiento abstracto de los objetos de la aplicación. Las operaciones pueden devolver excepciones (condiciones de error) que se almacenan en la metaclase `Exception`.

3.4.5 ODL. El lenguaje de definición de objetos de ODMG

El estándar ODMG 3.0 cuenta con dos lenguajes de especificación de objetos, el Lenguaje de Definición de Objetos (ODL) y el Formato de Intercambio de Objetos

(OIF). ODL es un lenguaje de especificación utilizado para la definición de esquemas en ODMG. OIF es un lenguaje de especificación utilizado para cargar o descargar el estado actual de una base de datos desde o a un conjunto de archivos. En esta sección vamos a resumir el ODL.

ODL es un lenguaje de definición para la especificación de objetos. Los sistemas de gestión de bases de datos tradicionales permiten la definición de datos mediante herramientas de definición o bien mediante un lenguaje de definición de datos (DDL, *Data Definition Language*) [Elma97]. El DDL permite a los usuarios definir el esquema de una base de datos. ODL es un DDL para objetos, de forma que permite la definición de esquemas para bases de datos orientadas a objetos, concretamente para bases de datos ODMG. ODL define las características de los tipos, incluyendo las propiedades y las operaciones. Sin embargo, respecto a las operaciones, ODL sólo define sus signaturas, pero no realiza la definición de los métodos que implementan dichas operaciones.

Los esquemas de una base de datos ODMG forman parte de una declaración `module` y contienen las distintas interfaces y clases que forman el esquema. Además, se indican las distintas excepciones que pueden provocar las operaciones incluidas en las definiciones de las interfaces y de las clases. La especificación de una interfaz en ODL define las propiedades y operaciones, así como la lista de interfaces que actúan como supertipo (superinterfaz) de la interfaz que se está definiendo. La especificación de una clase en ODL define las propiedades y operaciones que forman la definición de la clase, y opcionalmente, puede incluir una lista de las interfaces en las que se basa la definición de la clase. Además, las definiciones de clase pueden especificar un único nombre de clase que indica la superclase a la que se extiende. Por último, en la definición de una clase también se puede especificar si se quiere dejar encargado al sistema la gestión del conjunto de instancias de la clase mediante la inclusión de una cláusula `extent`. A continuación se resumen brevemente las partes más importantes de la gramática de ODL.

3.4.5.1 Módulos, clases, interfaces y excepciones

Un esquema se define mediante un módulo. Un módulo incluye la definición de las clases e interfaces que lo componen, así como la especificación de las excepciones que se pueden producir durante la ejecución de las operaciones definidas en las clases e interfaces del módulo. A continuación se describe la sintaxis ODL en EBNF para la definición de módulos, excepciones, interfaces y clases.

```
<module> ::= module <identifier> { <specification> }
<specification> ::= <definition> | <definition> <specification>
<definition> ::= <type_dcl>; | <const_dcl>; | <except_dcl>; | <interface>; |
        <module>; | <class>;

<except_dcl> ::= exception <identifier> { [<member_list>] }
<interface> ::= <interface_dcl> | <interface_forward_dcl>
<interface_dcl> ::= <interface_header> { <interface_body> }
<interface_forward_dcl> ::= interface <identifier>
```

```
<interface_header> ::= interface <identifier> [ <inheritance_spec> ]
<interface_body> ::= <export> | <export> <interface_body>
<export> ::= <type_dcl>; | <const_dcl>; | <except_dcl>; | <attr_dcl>; |
        <rel_dcl>; | <op_dcl>;

<inheritance_spec> ::= <scoped_name> | [ , <inheritance_spec>]
<class> ::= <class_dcl> | <class_forward_dcl>
<class_dcl> ::= <class_header> { <interface_body> }
<class_forward_dcl> ::= class <identifier>
<class_header> ::= class <identifier>
        [ extends <scoped_name> ]
        [ <inheritance_spec> ]
        [ <type_property_list> ]
<type_property_list> ::= ( [<extent_spec>] [<key_spec>] )
<extent_spec> ::= extent <string>
<key_spec> ::= key[s] <key_list>
<key_list> ::= <key> | <key> , <key_list>
<key> ::= <property_name> | ( <property_list> )
<property_name> ::= <identifier>
<property_list> ::= <property_name> | <property_name>, <property_list>
```

Atendiendo a esta gramática, un ejemplo sencillo de la definición de un módulo podría ser el siguiente:

```
module Example {
        interface Worker
        {
          properties
          operations
        };
        class Employee: Worker
        (extent theEmployees)
        {
          properties
          operations
        };
        class Client
        (extent theClients)
        {
          properties
          operations
        };
};
```

Este ejemplo define un módulo denominado `Example` formado por una interfaz `Worker` y dos clases: `Employee` y `Client`. La clase `Employee` está definida a partir de la interfaz `Worker`, por lo que hereda todas las definiciones realizadas en `Worker`. Además, las dos clases especifican su extensión para que sea el propio sistema el que se encargue de la gestión de sus respectivos conjuntos de instancias. Como aún no se ha visto cómo se especifican propiedades y operaciones, se ha dejado indicada la definición de las propiedades y operaciones de la interfaz y de las clases del módulo.

3.4.5.2 Atributos, relaciones y operaciones

Las propiedades de las instancias de un tipo se pueden definir mediante atributos o mediante relaciones; las operaciones se especifican indicando únicamente el nombre de la operación, el tipo devuelto y la lista de parámetros, indicando su tipo y si son de entrada, salida, o entrada y salida. A continuación se describe la sintaxis ODL en EBNF para la definición de propiedades y operaciones.

```
<attr_dcl> ::= [readonly] attribute <domain_type> <attr_list>
<rel_dcl> ::= relationship <target_of_path> <identifier> inverse
        <inverse_traversal_path>
<target_of_path> ::= <identifier> | <coll_spec> < <identifier> >
<inverse_traversal_path> ::= <identifier> :: <identifier>
<op_dcl> ::= [<op_attribute>] <op_type_spec> <identifier> <parameter_dcls>
        [<raises_expr>] [<context_expr>]
<op_type_spec> ::= <simple_type_spec> | void
<parameter_dcls> ::= ( [<param_dcl_list>] )
<param_dcl_list> ::= <param_dcl> | <param_dcl> , <param_dcl_list>
<param_dcl> ::= <param_attribute> <simple_type_spec> <declarator>
<param_attribute> ::= in | out | inout
<raises_expr> ::= raises ( <scoped_name_list> )
```

A continuación se muestra un ejemplo de lo que podría ser la definición de la interfaz `Worker`, que a modo de ejemplo incluye dos atributos que representan la categoría y el sueldo de los trabajadores.

```
Interface Worker
{
  attribute string SSNumber;
  attribute long department;
};
```

3.4.5.3 Definición de un esquema en ODL

Una vez descrita la sintaxis básica para la definición de esquemas, a continuación se ilustra en ODL la definición del esquema propuesto en la 3.4.3 para una base de datos de personas.

```
module PeopleSchema {
        class Car
        (extent theCars)
        {
          attribute string numberPlate;
          attribute string trademark;
          attribute string model;
          relationship People belongsTo inverse People::has;
        };
        class People
        (extent thePeople)
        {
          attribute string DNI;
          attribute string name;
          relationship set<Car> has inverse Car::belongsTo;
        };
```

```
        class Client extends People
        (extent theClients)
        {
        };
        interface Worker
        {
          attribute string SSNumber;
          attribute string department;
        };
        class Employee extends People: Worker
        (extent theEmployees)
          attribute string category;
          attribute long salary;
        {
        };
        class Temporary: Worker
        (extent theTemporaries)
        {
        };
};
```

3.4.6 Productos compatibles con ODMG

En las secciones anteriores se ha realizado un resumen de la evolución que han sufrido las especificaciones de ODMG, así como de los distintos componentes del estándar ODMG como son su modelo de objetos, los metadatos, el lenguaje de definición de objetos y el lenguaje de consulta de objetos. En esta sección veremos en qué medida se ajustan los principales productos existentes en el mercado a cada uno de estos componentes. Para ello, realizaremos una pequeña descripción de los productos más importantes y resumiremos sus características en una tabla de compatibilidades. Los productos estudiados son los de eXcelon, Objectivity, Poet, Sun Microsystems, Computer Associates, Micro Data Base Systems y Versant.

- eXcelon Corporation es el fabricante de ObjectStore, el servidor de bases de datos de objetos más utilizado. ObjectStore soporta completamente el estándar ODMG; de hecho, fue el primer producto que permitía la creación de una base de datos Java compatible con ODMG.
- Objectivity es el fabricante de Objectivity for Java y de Objectivity/C++. Ambos productos son compatibles son su correspondiente ligadura de ODMG. La definición de esquemas se realiza a través de archivos de cabecera incluidos en el código de la aplicación, y posteriormente son procesados por un preprocesador de DDL que crea todas las clases necesarias para la definición del esquema de la base de datos.
- Poet Software ha sido una de las compañías más implicadas en la definición de ODMG, por lo que sus productos tienen un gran grado de compatibilidad con las especificaciones de ODMG. Su producto se denomina FastObjects (antes POET Object Server) e incluye un soporte para OQL, así como para las ligaduras de Java y C++.

- Sun Microsystems apoya la ligadura Java de ODMG y ha desarrollado el producto Java Blend que implementa la ligadura Java sobre bases de datos relacionales. Este producto ofrece una visión orientada a objetos de datos relacionales permitiendo a los programadores guardar objetos Java en una base de datos relacional sin la necesidad de crear código adicional para resolver el problema de la falta de correspondencia (*impedance mismatch*). Para ello utiliza un modelo de programación basado en la *persistencia transparente* que guarda de forma transparente y automática los objetos que se utilizan en la aplicación Java.
- Computer Associates es el fabricante de Jasmine, una base de datos orientada a objetos pura que incluye un entorno de desarrollo integrado y características para la implantación en varias plataformas. Jasmine es compatible con la ligadura de Java de ODMG y ofrece soporte para consultas OQL.
- Micro Data Base Systems dispone de Titanium como un producto que implementa el estándar ODMG incorporando un lenguaje de definición de objetos para la definición de esquemas y una ligadura para C++.
- Versant ofrece su producto Versant ODBMS que es una base de datos que implementa ODMG para la gestión de objetos o componentes complejos en entornos distribuidos con un alto grado de concurrencia. Dispone de interfaces para C++ y Java que siguen las especificaciones de sus respectivas ligaduras de ODMG.

La tabla siguiente resume las características de ODMG que implementa cada uno de los productos descritos.

Producto	ODL	OQL	Ligadura C++	Ligadura Java
ObjectStore				X
Objectivity	Parcial		Parcial	X
FastObjects		X	X	X
Java Blend		X		X
Jasmine		X		X
Titanium	X		X	
Versant			X	X

Tabla 4. Compatibilidad de algunos productos con el estándar ODMG

3.5 Conclusiones

Las bases de datos de tiempo real, las bases de datos temporles y los modelos temporales, tanto los que extienden ODMG como los que parten de otros modelos, nos proporcionan los conceptos necesarios para la construcción de almacenes de datos. Estos conceptos son utilizados en todas las fases del desarrollo del almacén, primero en los metadatos temporales de las fuentes de datos, en el enriquecimiento del esquema de las

fuentes de datos, en el modelo canónico de datos, en el esquema del propio almacén, y en una fase posterior de refresco y mantenimiento del mismo.

Por otra parte la definición de un estándar como el de ODMG viene a cubrir la carencia que siempre han tenido las bases de datos orientadas a objetos con relación a la falta de un estándar para el modelo, la definición y consulta de objetos. Esta carencia ha supuesto una limitación en cuanto a la portabilidad de aplicaciones que utilizaban bases de datos orientadas a objetos ya que, hasta la aparición de este estándar, no existía una especificación común que permitiese realizar de forma normalizada las operaciones más usuales.

En este capítulo se han presentado los principales conceptos temporales que serán utilizados en esta tesis. De forma breve se han presentado las BD de tiempo real y temporales y las cuestiones que pueden ser utilizadas en el proceso de construcción del almacén. También se han revisado los modelos de datos temporales más relavantes y se ha propuesto un modelos de tiempo con los elementos que se han considerado necesarios para nuestro trabajo. Por último se ha presentado el standard ODMG, que hemos elegido como modelo canónico de datos para nuestra arquitectura.

Capítulo 4

ENRIQUECIMIENTO DE ODMG CON ELEMENTOS TEMPORALES

Las especificaciones actuales del estándar ODMG no incluyen soporte explícito para el tratamiento de elementos temporales. La inclusión de esta funcionalidad en este estándar nos permitirá definir esquemas con características temporales que serán de utilidad a la hora de realizar la integración temporal de datos provenientes de diferentes fuentes. Con la ampliación será posible definir tanto los *esquemas temporales* de las fuentes de datos como el del almacén de datos (que en sí mismo ya tiene connotaciones temporales).

En este capítulo se estudia también como, una vez enriquecido ODMG con elementos temporales, se puede utilizar la definición de esquemas externos propuesta en trabajos anteriores del grupo para definir que parte del esquema de las fuentes será exportada para su posible incorporación al esquema del almacén de datos.

Este capítulo está estructurado como sigue. La sección 4.1 introduce el problema a resolver y los diferentes niveles de esquemas. En la sección 4.2 se presenta la extensión de ODMG propuesta. En la sección 4.3 se presenta un ejemplo a nivel de esquemas de datos. La sección 4.4 explica como se añaden la extensiones propuestas al ODL de ODMG. La sección 4.5 muestra el código ODL del ejemplo de la sección 4.3. Por último, la Sección 4.6 resume las conclusiones de este capítulo.

4.1 Introducción

En el Capítulo anterior estudiamos las principales extensiones existentes para la extensión de ODMG y añadirle funcionalidades temporales [Bert96], [Bert98], [Kako96], [Duma04], [Grif04]. La mayor parte de dichas extensiones están pensadas para ser utilizadas en Bases de Datos Temporales. Lo que nosotros necesitamos es poder definir esquemas que contengan objetos (clases, interfaces y atributos) que tengan características de tiempo que sean diferentes de otros objetos que no tengan dichas características, por tanto la semántica de dichos objetos será diferente. Por otra parte, también es necesario distinguir

entre el esquema de las fuentes de datos y el esquema del almacén. Los esquemas están formados por una jerarquía de clases e interfaces. Las interfaces definen el comportamiento común entre clases que no tienen por que estar relacionadas mediante herencia (p.e. dos clases pueden compartir varias operaciones y no tener en común ninguna de sus instancias).

Aunque la extensión de ODMG se haga de forma genérica, es posible que determinadas características solo tenga sentido contemplarla en el almacén de datos. En la figura 1 podemos ver la relación entre los diferentes esquemas, los procesadores que se encargan de la conversión de un esquema en el siguiente y en que modelo están expresados dichos esquemas. El esquema nativo está expresado en el Modelo nativo de la fuente que puede ser cualquiera o incluso no tener ningún esquema. El procesador de Conversión se encargaría de traducir el esquema nativo al esquema componente expresado en ODMG+T. Nosotros partimos de dicho esquema componente, es decir, ya se ha realizado la fase de enriquecimiento semántico desarrollada en tesis anteriores [Cast93]. En el capítulo 2 se ha presentado la arquitectura de partida en la que se explica el proceso a seguir. De los esquemas componentes se puede pasar a los esquemas de exportación, tarea que realiza el procesador de Negociación. Los esquemas de exportación, expresados en ODMG+T, son esquemas externos del esquema componente. La idea es que cada uno de los esquemas componentes "exporte" solo la parte que sea de utilidad para su incorporación al Almacén de Datos y "esconda" la parte que por otros motivos (seguridad, privacidad, etc.) no sea necesario para formar el esquema del almacén de datos.

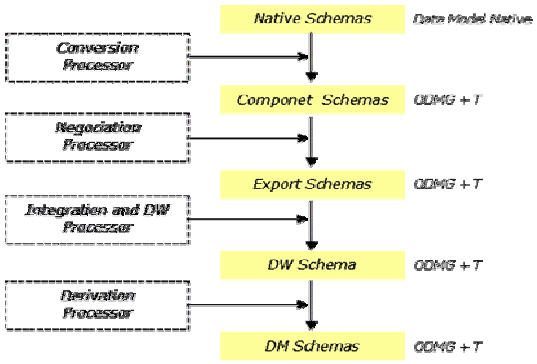

Figura 25. Procesadores, esquemas y ODMG

Para generar los esquemas de exportación utilizamos el mecanismo propuesto en [Torr00], [Torr01a], [Torr01b] para generar esquemas externos. En el mecanismo de definición de esquemas externos (nuestros esquemas de exportación) propuesto, los esquemas de usuario sólo utilizan las relaciones permitidas en el paradigma de orientación a objetos (herencia, agregación y asociación). Sin embargo, como las clases derivadas están relacionadas mediante derivación en el repositorio, cuando éstas sean incluidas en algún esquema externo (de exportación) se tendrán que obtener las relaciones de herencia

existentes con el resto de clases del esquema. Por tanto, se trata de un mecanismo definido sobre un modelo de objetos estándar que resuelve el problema de la integración de las clases derivadas de forma sencilla sin requerir la extensión del paradigma de orientación a objetos.

El mecanismo de definición de esquemas externos está basado en el propuesto en [Samo97] pero ha sido adaptado para el estándar ODMG. Por tanto, las clases derivadas son definidas e integradas igualmente en el repositorio. La integración se realiza utilizando la *relación de derivación* [Bert92], que relaciona en el repositorio las clases derivadas con sus clases base, y cuya semántica es "se ha definido (*derivado*) a partir de". De esta manera se resuelve de forma sencilla el problema de la integración de las clases derivadas sin necesidad de generar clases intermedias [Scho91b] [Rund92a]. Sin embargo, a diferencia de otros mecanismos definidos sobre ODMG que también utilizan esta relación [Garc00], ésta no es utilizada fuera del repositorio, por lo que los esquemas externos siguen siendo esquemas orientados a objetos convencionales. No obstante, se podría pensar que la utilización de la relación de derivación en el repositorio constituye una extensión del paradigma de orientación a objetos, pero esto no es cierto ya que el esquema del repositorio (*metaesquema*) sigue siendo un esquema orientado a objetos [Torr01c].

4.2 Extensión de ODMG

A continuación se expone la extensión propuesta para ODMG en la que se incluyen los elementos necesarios para contemplar las propiedades temporales necesarias.

4.2.1 Extensión de la jerarquía de tipos

En el capítulo 3 se explicaron los elementos temporales necesarios. Lo que proponemos es extender la jerarquía de tipos de ODMG para incorporar nuevos tipos con semántica temporal. La extensión propuesta no provee de tipos de objeto directamente instanciables. En lugar de ello proporciona una serie de tipos con características y semántica temporal que pueden ser incluidos utilizados en el enriquecimiento semántico de las fuentes y por el usuario para definir esquemas. Estos tipos de datos pueden ser estáticos (correspondientes al conjunto básico de tipos de ODMG), u objetos de tipo temporal. Nuestro modelo contiene el conjunto de todos los tipos de ODMG y los extiende añadiendo nuevos tipos de datos con semántica temporal. Los nuevos tipos serían:

- *Instante (instant)*: la continuidad del tiempo puede modelarse como una línea recta formada por un conjunto infinito de puntos Ti denominados instantes. Es una colección totalmente ordenada mediante la relación '≤'.
- Un *marca de tiempo (timestamp)* es un valor de tiempo asociado con un objeto, por ejemplo un atributo o una tupla.

- *Unidad temporal (temporal unit)*: es una partición del tiempo en un conjunto de intervalos de tiempo disjuntos donde cada intervalo es de granularidad atómica. Por ejemplo: día, mes, año.
- *Intervalo (period):* una sección de la línea de tiempo, determinada por la duración entre dos instantes.
- *Evento (event)*: es un hecho que se produce en la realidad y que provoca un cambio en el estado de, al menos, una fuente de datos. Un evento tiene lugar en un punto particular de la línea de tiempo Ti y, por tanto, no tiene duración.
- *Tiempo de vida (lifespan)*: es el tiempo que un objeto está definido. El tiempo de validez de vida de un objeto corresponde con el periodo de tiempo en el cual el objeto existe en la realidad modelada. El tiempo de transacción de vida un objeto corresponde con el tiempo en el cual el objeto está grabado en el sistema operacional.
- *Tiempo de transacción (Transaction Time, TT)*: es el intervalo de tiempo durante el cual, un hecho esta disponible en las fuentes de datos.
- *Tiempo de validez (Valid Time, VT)*: corresponde con el intervalo de tiempo en el que un hecho es cierto en la realidad que se modela. Un determinado evento genera un intervalo de validez y un intervalo de transacción. Generalmente VT es proporcionado por el usuario.
- *Granularidad (granularity)*: nivel de detalle con que son marcados los datos. Por ejemplo, si tenemos una granularidad día, implica que tenemos una snapshot por registro y día.

4.2.2 Extensión con nuevos elementos

Extendemos ODMG con nuevos elementos temporales que permiten el manejo de manera uniforme tanto de los objetos temporales como de los que no lo son. El conjunto de tipos básicos de ODMG se amplia con nuevos tipos de datos con semántica temporal. Se crean un nuevo elemento, llamado *atributo temporal* que hereda su comportamiento base de sus respectivos tipos de datos del modelo de datos ODMG. Este nuevo tipo de datos permite incluir aspectos temporales a nivel de atributo. También se extienden las características de los tipos de datos class e interface del modelo ODMG, dando lugar a los tipos de datos *clases temporales* e *interfaces temporales*, pues el esquema del sistema se conforma mediante las relaciones entre clases e interfaces (herencia, composición,…) y es necesario poder incluir información temporal a nivel de esquema.

El modelo propuesto adopta el marcado temporal de cada uno de los atributos de los objetos para permitir cambios asíncronos y diferentes granularidades. Podemos hacer marcado temporal a nivel de atributo, clase e interface. Un subtipo puede redefinir las propiedades y operaciones que hereda para especializarlas según el comportamiento temporal y el rango de valores apropiados de las instancias del subtipo.

4.3 Ejemplo ilustrativo

A continuación se presenta un ejemplo en el que a partir de dos fuentes de datos se obtiene el almacén de datos (figura 2).

Partimos del esquema componente de cada fuente (suponemos que ya se ha realizado la fase de enriquecimiento semántico y se ha obtenido el esquema componente a partir de esquema nativo de la fuente). A partir de cada esquema componente se genera un esquema de exportación según los requerimientos del administrador de cada fuente. A partir de los esquemas de exportación se genera el esquema del almacén de datos. Este proceso es semiautomático y puede requerir la intervención del administrador del almacén de datos durante el proceso de integración.

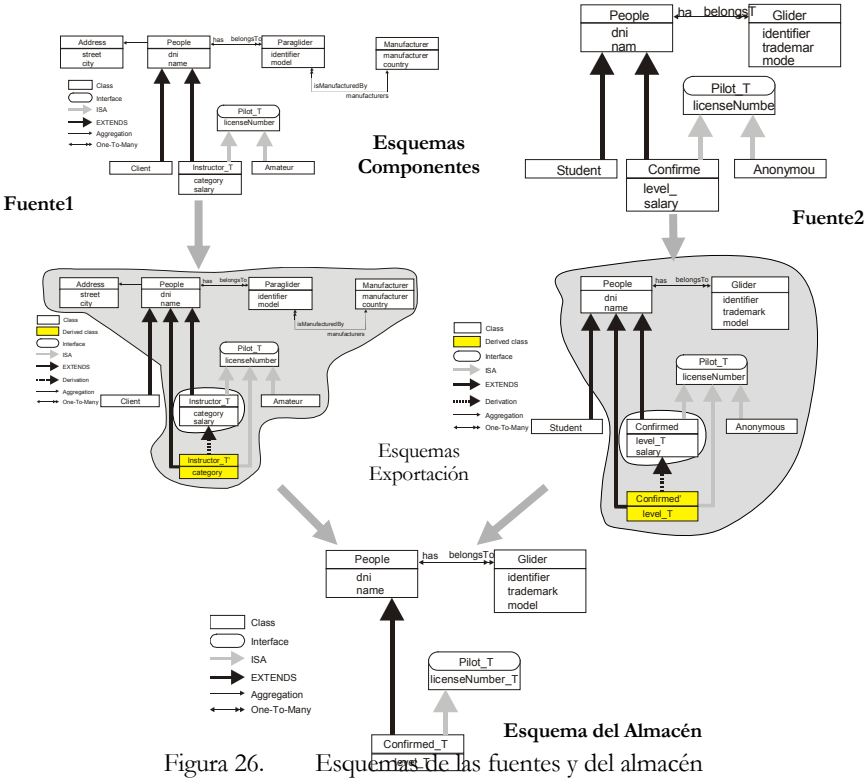

Figura 26. Esquemas de las fuentes y del almacén

En cualquier caso al final de la fase de integración el administrador validará el resultado obtenido. Como se puede ver en la figura hay clases, interfaces y atributos de ODMG y clases, interfaces y atributos temporales, que son con los que se ha extendido ODMG.

4.3.1 Esquema componente (fuente de datos 1)

La figura 3 muestra un esquema para una base de datos ODMG. Este esquema sirve de base para ilustrar el funcionamiento del mecanismo de definición de esquemas externos propuesto en [Torr02a]. Se trata de un esquema para una base de datos de pilotos de

vuelo libre. En concreto, podría corresponder con una de las federaciones autonómicas de parapente. Las personas pueden ser clientes o instructores. Las personas tienen parapentes, que están fabricados por una empresa de parapentes; un parapente sólo tiene un propietario. Además, las direcciones de las personas se almacenan como objetos. Los clientes tienen las mismas características que las personas; los instructores tienen algunas características adicionales. Además, se guarda información sobre los pilotos que no se dedican profesionalmente al parapente (amateurs). Los pilotos no profesionales y los instructores tienen propiedades y comportamiento en común, pero son diferentes de las características comunes entre los instructores y los clientes. La clase Instructor está definida a partir de la interfaz Piloto, por lo que hereda todas las definiciones realizadas en Piloto. Además, las dos clases especifican su extensión para que sea el propio sistema el que se encargue de la gestión de sus respectivos conjuntos de instancias. Para simplificar la figura, no se muestran las posibles operaciones que pudieran tener los componentes de este esquema.

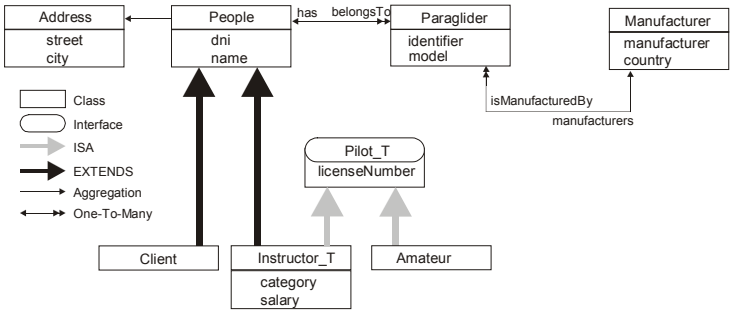

Figura 27. Esquema Conceptual de la Fuente1

4.3.2 Esquema de exportación (fuente de datos 1)

Para ilustrar la ventaja que ofrece este mecanismo de integración respecto a los mecanismos que proponen la integración de las clases derivadas utilizando únicamente relaciones de herencia [Scho91b, Rund92a], supongamos que queremos definir un esquema de exportación (esquema externo) a partir del esquema conceptual anterior sustituyendo la clase Instructor por una clase derivada Instructor'.

Instructor' oculta el sueldo de los instructores, ya que el administrador de la base de datos de la federación de parapente estima que no es una información de carácter público y no debe darse a conocer. La Figura 4 ilustra el repositorio una vez definida e integrada la clase derivada Instructor'.

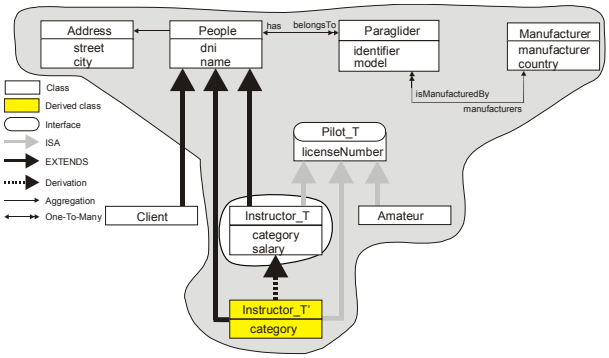

Figura 28. Esquema de Exportación de la Fuente1

4.3.3 Esquema componente (fuente de datos 2)

Para poder ilustrar el problema de la integración temporal es necesario introducir una segunda fuente de datos. Para el ejemplo se ha utilizado una base de datos empleada en un sistema de ayuda a la toma de decisiones para mejorar la calidad de los vuelos de los pilotos de vuelo sin motor, que funciona a través de Internet.

Todas las personas realizan sus vuelos con un determinado modelo de parapente. De estas personas nos interesan algunas de sus características, que deben facilitar a la hora de enviar un vuelo. Si los pilotos no desean facilitar sus características personales se englobarían dentro de la clase 'anónimo'.

Cuando las personas aun se encuentran en la fase de aprendizaje no vamos a tener en cuenta los vuelos que hayan podido realizar, puesto que éstos se verán mucho mas condicionados por las habilidades que va adquiriendo el alumno que por las características de la atmósfera. Cuando los alumnos finalizan su etapa de aprendizaje se les considera pilotos confirmados y se les supone suficiente capacidad para aprovechar las posibilidades de la atmósfera, teniendo en cuenta sus resultados a la hora de realizar los análisis.

Los pilotos confirmados van adquiriendo cada vez mayor experiencia y presentándose a diferentes pruebas pueden ir aumentando de nivel. Puesto que la licencia federativa de los pilotos se renueva cada año, podremos asociar un nivel determinado a cada piloto en este intervalo. Cuanto mayor sea el nivel de un piloto se le podrá considerar con mayor habilidad para aprovechar las posibilidades de la atmósfera.

Para ilustrar nuestro mecanismo de definición de esquemas de exportación (esquemas externos) supongamos que tenemos una base de datos de personas *que vuelan en parapente* con un esquema como el que se muestra en la Figura 5. Las personas pueden ser *alumnos* o *pilotos confirmados*, y ambos poseen *parapentes*. Los *alumnos* tienen las mismas características que las personas, pero sobre los *pilotos confirmados* se guardan algunas características más. Por cuestiones de estadística se introduce una variable en el

formulario de registro de los usuarios para que introduzcan el rango de su salario en la actividad profesional que realicen. En este caso, se trata de una información que se introduce una única vez cuando los usuarios se registran, por lo que su valor no tiene por qué modificarse. Por este motivo sólo se marca como temporal el atributo nivel, a diferencia de la fuente anterior donde se marcaba la clase completa como temporal. También se guarda información sobre los *pilotos anónimos*. Los *pilotos anónimos* y los *confirmados* tienen propiedades y comportamiento en común, pero diferentes de los rasgos comunes entre *confirmados* y *alumnos*. Para simplificar, en la figura no se incluyen ni atributos ni operaciones.

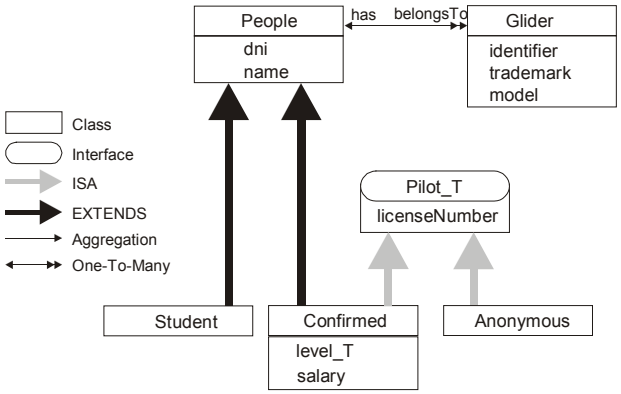

Figura 29. Esquema Componente de la Fuente2

4.3.4 Esquema de exportación (fuente de datos 2)

A la hora de diseñar el almacén el administrador entiende que el salario de los pilotos no es un dato necesario dado el tipo de consultas que se van a realizar sobre éste. Se crea entonces una clase derivada Confirmado' que oculta esta información. No obstante, esta metodología hace este problema transparente al usuario, ya que la nueva interfaz es generada por el sistema.

La Figura 6 ilustra el repositorio con las clases e interfaces derivadas coloreadas e integradas mediante derivación. El esquema de exportación (esquema externo) está representado por el área sombreada. Obsérvese como la relación de derivación se utiliza en el repositorio pero no en el esquema de exportación.

Para ilustrar nuestro mecanismo de definición de esquemas de exportación supongamos que tenemos una base de datos de personas *que vuelan en parapente* con un esquema como el que se muestra en la Figura 6. Las personas pueden ser *estudiantes* o *pilotos confirmados*, y ambos poseen *parapentes*. Los *alumnos* tienen las mismas características que las personas, pero sobre los *pilotos confirmados* se guardan algunas características más. También se guarda información sobre los *pilotos anónimos*. Los *pilotos anónimos* y los *confirmados* tienen propiedades y comportamiento en común, pero diferentes de los rasgos comunes entre

confirmados y *estudiantes*. Para simplificar, en la figura no se incluyen ni atributos ni operaciones.

Supongamos que queremos definir un esquema externo a partir del esquema conceptual anterior que oculte cierta característica de los *pilotos confirmados, por ejemplo el salario personal del piloto (el salario del lugar donde trabaja el piloto; se supone que como piloto no cobra nada, solo es una afición)* A la hora de diseñar el almacén el administrador entiende que el salario de los pilotos no es un dato necesario dado el tipo de consultas que se van a realizar sobre éste. Se crea entonces una clase derivada Confirmado' que oculta esta información. No obstante, esta metodología hace este problema transparente al usuario, ya que la nueva interfaz es generada por el sistema.

La Figura 6 ilustra el repositorio con las clases e interfaces derivadas coloreadas e integradas mediante derivación. El esquema externo está representado por el área sombreada. Obsérvese como la relación de derivación se utiliza en el repositorio pero no en el esquema externo.

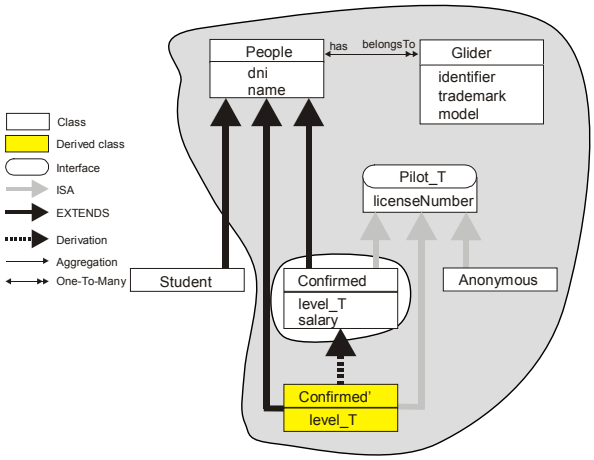

Figura 30. Esquema de Exportación de la Fuente2

4.3.5 Esquema del almacén de datos

El esquema del almacén, obtenido tras la integración de las dos fuentes de datos anteriores, se muestra en la figura 7. Se ha optado por un tipo de esquema orientado al almacenamiento, siguiendo la aproximación de [Inmo02]. No se reflejan en el esquema del almacén por tanto ni la tabla de hechos ni las diferentes dimensiones porque las consultas no se lanzarán directamente sobre el almacén, sino sobre los diferentes data mart a los que alimenta, por lo que deben ser estos los que deben utilizar un esquema adecuado a la consulta y no el almacén en sí.

Se ha resumido toda la información que se tenía sobre los diferentes tipos de pilotos en una sola clase, denominada Confirmado. Lo mismo ha ocurrido con la clase Planeador, que resume toda la información disponible en ambas fuentes de datos sobre los

parapentes. Puede observarse como no aparece en el esquema resultante ninguna información acerca del salario de los usuarios, puesto que este esquema se construye a partir del esquema de exportación de las fuentes, y en ninguno de los dos esta información estaba disponible.

En el capítulo 6 se detallan los algoritmos que determinan la granularidad de algunos de los atributos de las clases del esquema del almacén (level granularity year).

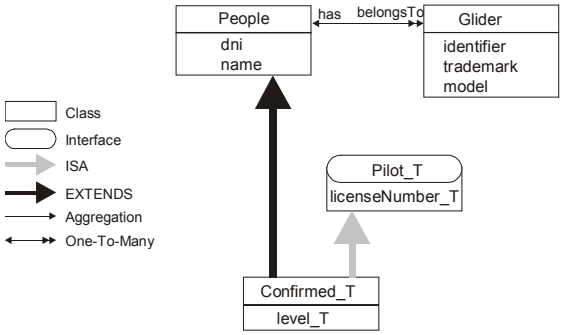

Figura 31. Esquema del AD formado a partir de los Esq. de exportación de las fuentes

4.3.6 Mapeo de las fuentes al almacén de datos

En la figura 8 se muestra gráficamente que atributos de las clases del esquema de las fuentes se integran entre sí, dando lugar a un atributo de alguna clase del esquema del almacén. Se han obviado aquellas que son evidentes para simplificar la figura, como el DNI o el nombre.

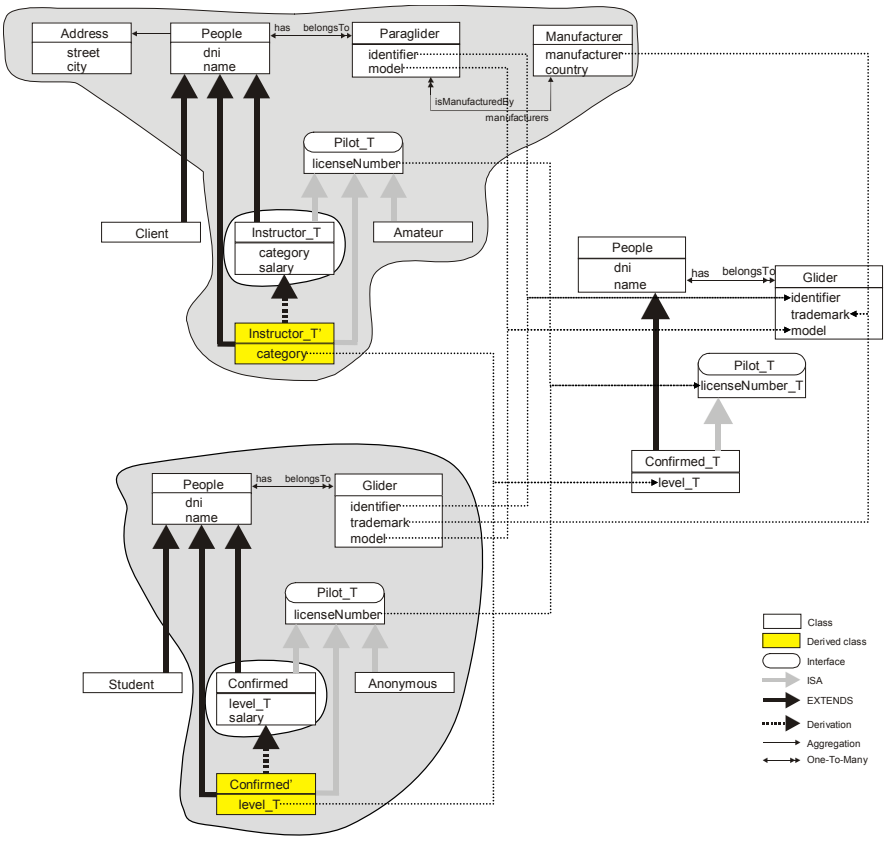

Figura 32. Esquemas de las fuentes y mapeo al almacén

4.4 Extensión de ODL

En esta sección vamos a resumir el ODL de forma que podamos definir los esquemas con la extensión de ODMG propuesta siguiendo la sintaxis de ODL.

ODL es un lenguaje de especificación utilizado para la definición de esquemas en ODMG. Es un lenguaje de definición para la especificación de objetos. Los sistemas de gestión de bases de datos tradicionales permiten la definición de datos mediante herramientas de definición o bien mediante un lenguaje de definición de datos (DDL, Data Definition Language) [Elma97]. El DDL permite a los usuarios definir el esquema de una base de datos. ODL es un DDL para objetos, de forma que permite la definición de esquemas para bases de datos orientadas a objetos, concretamente para bases de datos ODMG. ODL define las características de los tipos, incluyendo las propiedades y las operaciones. Sin embargo, respecto a las operaciones, ODL sólo define sus signaturas, pero no realiza la definición de los métodos que implementan dichas operaciones.

4.4.1 Clases e interfaces

Un esquema se define mediante un módulo. Un módulo incluye la definición de las clases e interfaces que lo componen, así como la especificación de las excepciones que se pueden producir durante la ejecución de las operaciones definidas en las clases e interfaces del módulo. A continuación se describe la sintaxis ODL en EBNF para la definición de interfaces y clases. Hemos añadido las extensión de ODMG que hemos propuesto en este capítulo para elementos temporales. En concreto se añade interface_T para las interfaces temporales y class_T para las clases temporales. A continuación mostramos la modificación, las incorporaciones se indican en cursiva y negrita.

```
<interface> ::= <interface_dcl> [<temp_desc>]
              | <interface_forward_dcl> [<temp_desc>]
<interface_dcl> ::= <interface_header> { <interface_body> }
<interface_forward_dcl> ::=   interface <identifier> |
                              interface_T <identifier>
<interface_header> ::= interface <identifier> [ <inheritance_spec> ] |
                       interface_T <identifier> [<inheritance_spec>]
<interface_body> ::= <export> | <export> <interface_body>


<class> ::= <class_dcl> [<temp_desc>]  | <class_forward_dcl> [<temp_desc>]
<class_dcl> ::= <class_header> { <interface_body> }
<class_forward_dcl> ::=       class <identifier> |
                              class_T <identifier> |
<class_header> ::=            class <identifier> |
                              class_T <identifier> |
       [ extends <scoped_name> ]
       [ <inheritance_spec> ]
       [ <type_property_list> ]

<temp_desc> ::= [ <TT_Description> ] [ <VT_Description> ]
<TT_Description> ::= [ TT <Temp_type> [Granularity <value>] ]
<VT_Description> ::= [ VT <Temp_type> [Granularity <value>] ]
<Temp_type> ::= [ Event|Interval ]
```

4.4.2 Atributos

Las propiedades de las instancias de un tipo se pueden definir mediante atributos o mediante relaciones; las operaciones se especifican indicando únicamente el nombre de la operación, el tipo devuelto y la lista de parámetros, indicando su tipo y si son de entrada, salida, o entrada y salida. A continuación se describe la sintaxis ODL en EBNF para la definición de atributos. Hemos añadido las extensión de ODMG que hemos propuesto en este capítulo para elementos temporales. En concreto se añade attribute_T para los atributos temporales.

```
<attr_dcl> ::= [readonly]      attribute <domain_type> <attr_list> |
                               attribute_T <domain_type> <attr_list>
```

4.5 Código en ODL del ejemplo

Una vez definidas las extensiones al EBNF para la definición de elementos temporal mostramos el código correspondiente a los esquemas componentes y de exportación de las dos fuentes de datos, y el código correspondiente al almacén de datos formado a partir de ambas fuentes.

A continuación se detalla el código necesario para crear el esquema anterior en una base de datos con soporte para ODMG+T. La mayoría de las definiciones de los elementos del esquema se corresponde exactamente con la definición del esquema que debería realizarse mediante el ODMG en su versión estándar. Los elementos temporales incluyen ciertas características que no pueden expresarse usando únicamente ODMG estándar. En el caso de la definición de la interfaz `Pilot_T`, por ejemplo, se indica que todos los objetos que hagan uso de esta interfaz tienen una marca temporal (expresado mediante la palabra clave `TT`) que indica el momento (`Event`) en que se registró en la fuente de datos algún cambio, con un nivel de detalle de día (`Day`). También se indica que estos objetos tienen algún atributo que representa el instante (`Event`) en el que es válida la información del piloto (en este caso la licencia) en el mundo real (`VT`). Los objetos de tipo `Instructor_T`, en cambio no proveen de información acerca de cuando son válidos en la realidad, pero podría aproximarse si fuese necesario mediante la utilización de la marca temporal que indica cuando se registró en la fuente de datos, de la que si disponen (`TT`), y que sabemos que tiene un nivel de detalle de hora (`Hour`). Puesto que ambos casos la información temporal se asocia a nivel de clase, todos los miembros que forman parte de la clase poseen estas mismas características temporales. En el caso de que sólo algunos de los miembros de la clase posean características temporales, o sean diferentes entre ellos, se puede aplicar el mismo estilo de definición de elementos temporales de forma individual a cada uno de los miembros de la clase, como puede verse en la definición del atributo `Level` de la clase `Confirmed` en los esquemas de datos de la segunda fuente.

4.5.1 Código en ODL para esquema componente de fuente 1

```
class Address
{
  attribute string street;
  attribute string city;
}
class People (extent thePeople)
{
  attribute string DNI;
  attribute string name;
  attribute Address address;
  relationship Set<Paraglide> has inverse Paraglider::belongsTo;
}
class Paraglider (extent theGliders)
{
  attribute string identifier;
  attribute string trademark;
  attribute string model;
```

```
  relationship People belongsTo inverse People::has;
  relationship Manufacturer isManufacturedBy inverse
                       Manufacturer::manufactures;
}
class Manufacturer
{
  attribute string manufacturer;
  attribute string country;
  relationship Set<Paraglider> manufactures inverse
                       Paraglider::isManufacturedBy;
}
interface_T Pilot_T TT Event Granularity Day VT Event
{
  attribute string licenseNumber;
  attribute_T string licenseDate; VT Event Granularity Day
}
class_T Instructor_T TT Event Granularity Day extends People: Pilot_T
(extent theInstructors)
{
  attribute string category;
  attribute long salary;
}
class Amateur: Pilot
(extent theAmateurs)
{
}
class Client extends People
{
}
```

4.5.2 Código en ODL para esquema de exportación de fuente 1

```
class Address
{
  attribute string street;
  attribute string city;
}
class People (extent thePeople)
{
  attribute string DNI;
  attribute string name;
  attribute Address address;
  relationship Set<Paraglide> has inverse Paraglider::belongsTo;
}
class Paraglider (extent theGliders)
{
  attribute string identifier;
  attribute string trademark;
  attribute string model;
  relationship People belongsTo inverse People::has;
  relationship Manufacturer isManufacturedBy inverse
                       Manufacturer::manufactures;
}
class Manufacturer
{
  attribute string manufacturer;
  attribute string country;
  relationship Set<Paraglider> manufactures inverse
```

```
                        Paraglider::isManufacturedBy;
}
interface_T Pilot_T TT Event Granularity Day VT Event
{
  attribute string licenseNumber;
}
class_T Instructor1_T TT Event Granularity Day virtual extends People: Pilot_T
(extent theInstructors)
{
  attribute string category;
}
class Amateur: Pilot
(extent theAmateurs)
{
}
class Student extends People
{
}
```

4.5.3 Código en ODL para esquema componente de fuente 2

```
class People (extent thePeople)
{
  attribute string DNI;
  attribute string name;
  relationship Set<Glider> has inverse Glider::belongsTo;
}
class Glider (extent theGliders)
{
  attribute string identifier;
  attribute string trademark;
  attribute string model;
  attribute string manufacturer;
  relationship People belongsTo inverse People::has;
}
interface_T Pilot_T TT Event Granularity Day VT Event
{
  attribute string licenseNumber;
}
class Confirmed extends People: Pilot_T
extent theConfirmeds)
{
  attribute_T string level TT Event Granularity Hour;
  attribute long salary;
}
class Anonymous: Pilot
(extent theAnonymous)
{
}
```

4.5.4 Código en ODL para esquema de exportación de fuente 2

```
class People (extent thePeople)
{
  attribute string DNI;
  attribute string name;
  relationship Set<Glider> has inverse Glider::belongsTo;
}
```

```
class Glider (extent theGliders)
{
  attribute string identifier;
  attribute string trademark;
  attribute string model;
  attribute string manufacturer;
  relationship People belongsTo inverse People::has;
}
interface_T TT Event Granularity Day VT Event Pilot_T
{
  attribute string licenseNumber;
}
class Confirmed1 virtual extends People: Pilot_T
extent theConfirmeds)
{
  attribute_T string level TT Event Granularity Hour;
}
class Anonymous: Pilot
(extent theAnonymous)
{
}
```

4.5.5 Código en ODL para esquema del almacén de datos

```
class People
( extent thePeople)
{
  attribute string DNI;
  attribute string name;
  relationship Set<Glider> has inverse Glider::belongsTo;
}
class Glider
( extent theGliders)
{
  attribute string identifier;
  attribute string trademark;
  attribute string model;
  relationship People belongsTo inverse People::has;
}
interface_T Pilot_T
{
  attribute_T string licenseNumber TT Event Granularity Day;
}
class_T Confirmed_T extends People: Pilot_T
extent theConfirmeds)
{
  Attribute_T string level TT Event Granularity Day;
}
```

4.6 Conclusiones

El enriquecimiento de ODMG con elementos temporales tiene como objetivo dar solución al problema de representar propiedades temporales (a parte de las no temporales que ya ofrece ODMG) en un esquema.

En este capítulo se ha propuesto una extensión de ODMG para incorporar elementos temporales, diferenciándolos de los no temporales. Del esquema nativo de la fuente se pasa al esquema componente a través de un proceso de enriquecimiento semántico. El esquema componente se transforma en el esquema de exportación ocultando las propiedades que se consideran necesarias. Tanto los esquemas componentes como los esquemas de exportación están enriquecidos con objetos temporales. Al incorporar nuevos objetos al modelo es necesario modificar la sintaxis ODL en EBNF para la definición de interfaces, clases y atributos con semántica temporal. A lo largo del capítulo se explica un ejemplo ilustrativo que muestra el proceso de conversión e integración de esquemas así como los esquemas correspondientes enriquecidos con elementos temporales.

Capítulo 5

EXTENSIÓN DE LOS METADATOS DE ODMG PARA CONTEMPLAR ELEMENTOS TEMPORALES

Los metadatos en una base de datos, entre otra información, representan información descriptiva acerca de los objetos que forman los distintos esquemas de la base de datos. En el estándar ODMG, dicha información se guarda en el *repositorio de esquemas*, cuya estructura (metaesquema) también es un esquema orientado a objetos. El metaesquema del estándar ODMG representa cuáles son los metadatos de ODMG, así como sus relaciones. Sin embargo, dado que las especificaciones de ODMG 3.0 no incluyen soporte para la definición de elementos temporales ni para la definición de las características temporales de las fuentes de datos, existen ciertas carencias en sus metadatos que pueden ser solucionadas mediante la extensión propuesta en este capítulo. Esta extensión es necesaria porque es obligado guardar información sobre las fuentes de datos para poder realizar la integración temporal de los datos, el refresco del almacén y el mantenimiento posterior del mismo. También será necesario disponer de elementos con propiedades temporales que nos permitan definir el esquema de las fuentes y el esquema del almacén. Algunos de las clases, interfaces y atributos del esquema tendrán propiedades temporales que es necesario representar de forma separada de las no temporales. La existencia de estos conceptos implica la definición de sus metadatos correspondientes y, en general, de todos los metadatos necesarios para dar soporte a lo antes mencionado.

El capítulo se estructura como sigue. En la sección 5.1 se presentan las principales características de los metadatos de ODMG. En la sección 5.2 se comentan las limitaciones de los metadatos que nos afectan para nuestro trabajo. En la sección 5.3 se proponen las extensiones de los metadatos necesarias. En la sección 5.4 se definen en ODL los metadatos propuestos. En la sección 5.5 se muestra un ejemplo ilustrativo. Y finalizamos con las conclusiones en la sección 5.6.

5.1 Objetos del metaesquema, objetos del esquema y objetos de datos

Los metadatos en una base de datos, entre otra información, representan información descriptiva acerca de los objetos que forman los distintos esquemas de la base de datos. En el estándar ODMG, dicha información se guarda en el *repositorio de esquemas*, cuya estructura (metaesquema) también es un esquema orientado a objetos. En esta sección describiremos los distintos tipos de objetos que se guardan en una base de datos. Utilizaremos esta descripción para describir la estructura del repositorio de esquemas de ODMG y proponer su extensión para la definición de esquemas externos.

Los objetos de una base de datos pueden ser de tres tipos dependiendo del nivel en que se encuentren: objetos del metaesquema, objetos del esquema y objetos de datos [Tres92], [Goer93].

- *Objetos del nivel metaesquema*. Son objetos que describen el metaesquema de la base de datos. Cada modelo de objetos ofrece una serie de metaobjetos que permiten la definición de esquemas de bases de datos. Dichos objetos se corresponden con los distintos tipos y conceptos que ofrece el modelo de objetos; se les suele denominar metaclases.
- *Objetos del nivel esquema*. Son objetos que describen el esquema de la aplicación, y son los que se utilizan en la definición del esquema conceptual y de los esquemas externos. Estos objetos son instancias de las metaclases (objetos del nivel metaesquema).
- *Objetos de datos*. Son los objetos que realmente guardan los usuarios en la base de datos. Son instancias de las clases definidas a nivel de esquema.

Los objetos se crean de forma descendente, de forma que inicialmente una base de datos sólo consta de objetos del nivel metaesquema. Cuando el administrador de la base de datos define el esquema conceptual y los esquemas externos, se crean los objetos del nivel esquema. Por último, es el usuario el que se encarga de crear los distintos objetos de datos que se guardan en la base de datos. La figura 5.1 ilustra un ejemplo de los distintos objetos que podemos encontrar en cada uno de estos niveles.

En el nivel metaesquema de la figura 5.1 se representa un fragmento del repositorio de esquemas de ODMG, mostrando las metaclases correspondientes a `Module`, `Type`, `Interface`, `Class`, `Property` y `Attribute`. Las instancias de estas metaclases son los distintos módulos, clases e interfaces definidas en los módulos, junto con las propiedades definidas en las clases e interfaces.

En el nivel esquema de la figura se ilustra una simplificación del esquema conceptual de un esquema relativo a una base de datos de personas, las cuales pueden ser clientes o empleados. Dicho esquema es instancia de la metaclase `Module`, lo que queda representado en la figura mediante la línea punteada que existe entre el recuadro sombreado que rodea a las clases del esquema y la metaclase `Module`. Las clases definidas

en este esquema son instancias de la metaclase `Class`, y los atributos definidos en las clases son instancias de la metaclase `Attribute`.

Por último, el nivel de objetos muestra distintos objetos de la base de datos como instancias de `Client` o `Employee`.

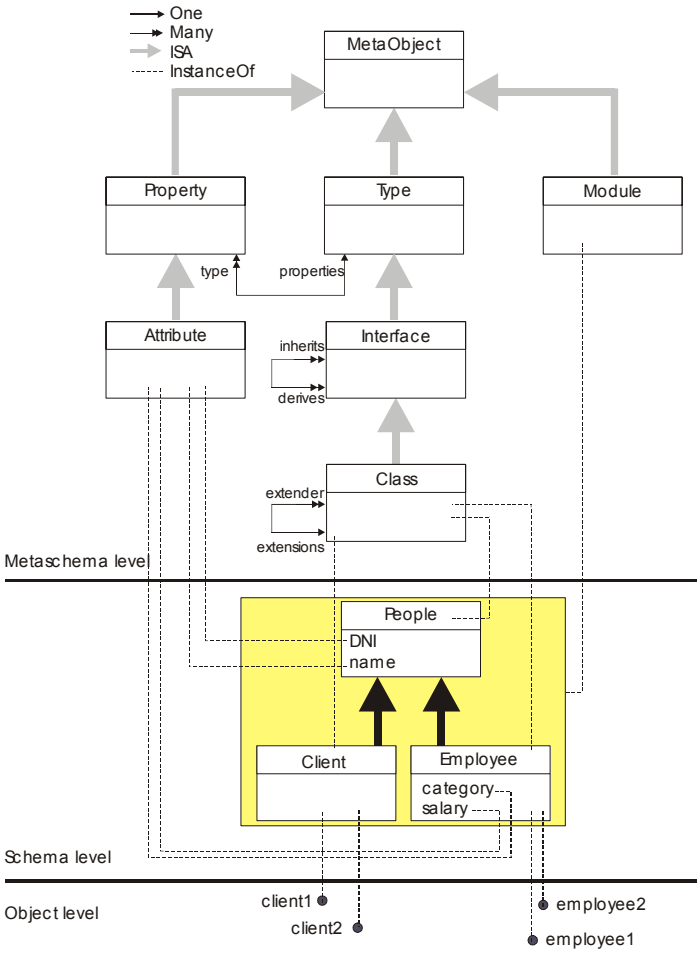

Figura 33. Objetos del metaesquema, objetos del esquema y objetos de datos

5.2 Limitaciones de los metadatos actuales de ODMG

Las especificaciones actuales de ODMG no permiten la definición de *elementos temporales* por lo que esta puede ser la razón por la que tampoco aparecen incluidas en dichas especificaciones las definiciones correspondientes a sus metadatos. No obstante, dichos metadatos podrían estar definidos en las especificaciones de ODMG independientemente de que existan o no los mecanismos para su definición, de forma que estableciesen las características estándar que deben incluir dichos metadatos para si fuera necesario desarrollar los mecanismos de definición apropiados.

Sin embargo, la extensión de los metadatos de ODMG para dar soporte a la definición de elementos temporales no se limita únicamente a definir los metadatos para dichos elementos, sino que también exige realizar ciertas modificaciones sobre algunas definiciones de metadatos existentes.

Esto se debe a que ODMG no considera la existencia varios tipos de esquemas, por lo que una base de datos ODMG está formada únicamente por un esquema, el esquema conceptual. De esta forma, si se permitiese la definición de otro tipo de esquemas utilizando los metadatos actuales de ODMG no se podría saber cuáles son las clases o interfaces incluidas en cada uno de los esquemas definidos, o cuál es la relación de herencia existente entre dos clases o interfaces en un esquema determinado. En las especificaciones actuales sólo se representan cuáles son las clases o interfaces que se han definido explícitamente en un esquema y qué relación existe entre clases (resp. interfaces) de forma genérica, pero sin tener en cuenta el esquema en que se produce. Concretamente, esta información se obtiene a partir del concepto de ámbito, que expresa el lugar de definición de cualquier componente de un esquema.

En nuestro caso nos interesa saber en que esquema se utiliza una determinada fuente de datos. Por otro lado también es importante saber que fuentes de datos son utilizadas en un determinado esquema. Puede darse el caso que un almacén de datos esté alimentado por varias fuentes de datos (y que esto quede reflejado en el esquema), pero también que dichas fuentes de datos alimenten a otro almacén.

En el Capítulo 3, dedicado al estándar ODMG, se describieron brevemente los principales metadatos de ODMG, como son `MetaObject`, `Scope`, `Module`, `Interface`, `Class`, `Attribute`, `Relationship`, `Operation` y `Exception`. Las instancias de `MetaObject` se corresponden con los metaobjetos que representan a los elementos del esquema guardados en el repositorio. En la metaclase correspondiente a `Scope` se guardan las instancias que representan a ámbitos de definición de otros metaobjetos. `Module` incluye una instancia para cada uno de los módulos definidos. `Interface` y `Class` contienen cada una de las definiciones de interfaces y `clases` realizadas, respectivamente. Análogamente, las metaclases asociadas a `Attribute` y `Relationship` guardan, respectivamente, la definición de cada uno de los atributos y relaciones realizada en alguna clase o interfaz. Por último, en `Operation` se guardan las operaciones definidas, y `Exception` incluye las instancias de cada una de las excepciones que pueden devolver las operaciones. La figura 5.2 ilustra la parte del metaesquema de ODMG.

5.2.1 Limitación para conocer los componentes de un módulo

Como se puede observar en la figura 5.2, todos los metadatos salvo `Scope` y `DefiningScope` están definidos directa o indirectamente a partir de `MetaObject`, por lo que heredan sus propiedades y sus operaciones. Como `MetaObject` tiene una relación con `DefiningScope` (la relación `definedIn`), todos los metaobjetos de la figura 5.2, salvo `Scope`

y `DefiningScope` tienen esta relación heredada de `MetaObject`. Es decir, todos los metaobjetos, excepto `Scope` y `DefiningScope` tienen un ámbito de definición, que indica el objeto en el que se han definido (p.e. una clase puede representar el ámbito de definición para el caso de una propiedad, o un módulo puede representar el ámbito de definición para el caso de una clase o de un submódulo). Esta relación es 1:M, lo que indica que una instancia de `MetaObject` sólo puede estar definida en un ámbito, y un ámbito puede contener varias instancias de `MetaObject` (p.e. una clase está definida en un módulo y un módulo puede contener la definición de varias clases). Esta relación indica el lugar de definición original de los metaobjetos, pero no indica en qué ámbitos se utilizan. Es decir, con la propuesta actual de los metadatos de ODMG sólo se puede saber en qué módulo se ha definido una clase, pero no en qué módulos se utiliza. Esto ocurre, como hemos comentado antes, porque ODMG no considera que se puedan definir esquemas a partir del esquema conceptual, por lo que una clase o una interfaz sólo pertenecen al esquema en el que se ha definido. En cambio, cuando pueden existir varios esquemas que utilicen las clases o interfaces definidas en otros esquemas existentes, es necesario modelar la relación de utilización. Asimismo, este mismo problema se da en las propiedades y relaciones de una clase, así como el resto de componentes de los módulos, clases e interfaces, que también necesitan modelar la relación de utilización.

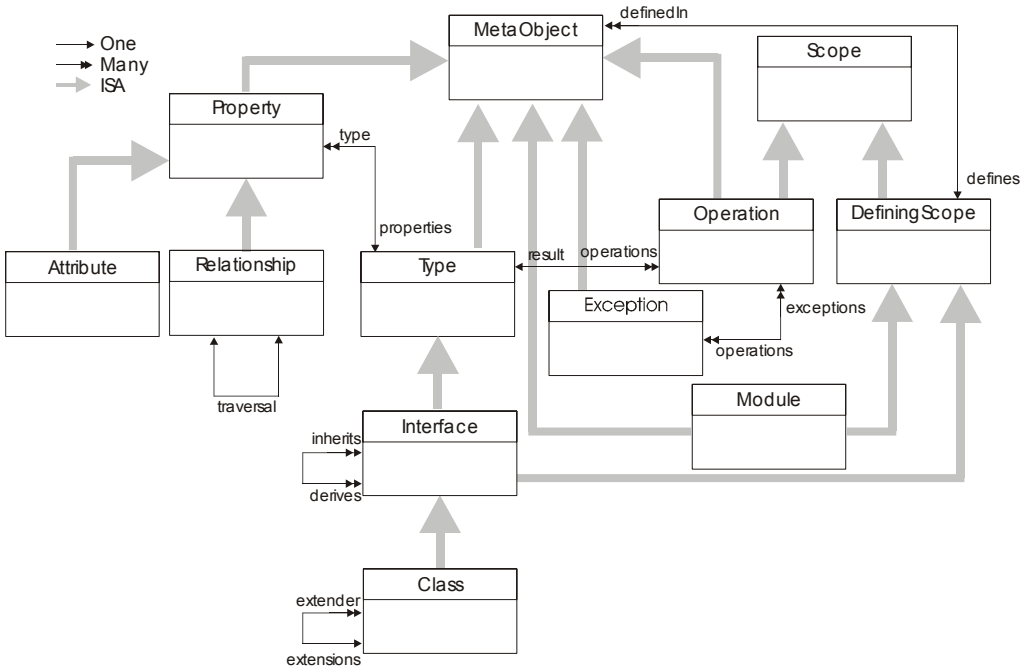

Figura 34. Fragmento del metaesquema de ODMG

No obstante, podríamos pensar que la relación de utilización es la propia relación `definedIn`, en lugar de una nueva relación. Sin embargo, la relación que necesitamos es una relación M:N en lugar de una 1:M, ya que un mismo metaobjeto puede estar incluido

en varios ámbitos (p.e. una misma clase puede estar incluida en varios esquemas y un esquema puede contener varias clases). Así pues, la relación `definedIn` hace referencia al lugar de definición de los metaobjetos, pero no a los distintos lugares en que se utilizan, por lo que es necesario modelar esta nueva relación.

Por tanto, es necesario realizar ciertas modificaciones sobre los metadatos actuales del estándar ODMG, de forma que se puedan saber qué instancias de `MetaObject` están incluidas en un ámbito determinado. Esto supone la modificación de los metadatos correspondientes a `MetaObject` y `DefinitionScope`.

5.2.2 Limitación para conocer las relaciones de herencia por esquema

Otra carencia que encontramos en el metaesquema de ODMG es que las relaciones de subtipo entre interfaces y las relaciones de subclase entre clases se hacen de forma genérica, independientemente del esquema en que tengan lugar. Con la especificación actual de ODMG, todas las instancias de la metaclase `Interface` tienen una relación `inherits-derives`, que indican, respectivamente, las superinterfaces y las subinterfaces de una interfaz. Esta relación es una relación M:N debido a la posibilidad de herencia múltiple entre interfaces. Con esta relación podemos saber cuáles son las superinterfaces o subinterfaces de una interfaz, pero no podemos saber en qué esquemas ocurren esas relaciones. Esta información no se puede obtener a partir de la definición del ámbito de la interfaz, ya que una interfaz se habrá definido en cierto esquema (módulo), que establecerá su ámbito de definición, pero no indica cuáles son todos los esquemas en los que se encuentra incluida. Asimismo, este problema también aparece en los metadatos correspondientes a `Class`, que no permiten obtener cuál es la relación de subclase existente entre dos clases en un esquema determinado. Los metadatos correspondientes a `Class` únicamente indican que posee una relación reflexiva, similar a la definida en `Interface`, denominada `extender-extensions`, que es una relación 1:M que establece de forma genérica la relación de herencia simple que permite el estándar ODMG para clases.

Por tanto, es necesario realizar ciertas modificaciones sobre los metadatos actuales del estándar ODMG, de forma que se puedan saber qué instancias de `MetaObject` están incluidas en un ámbito determinado, así como cuáles son las instancias de `Interface` y de `Class` junto con sus relaciones en una instancia de `Module`. Esto supone la modificación de los metadatos correspondientes a `MetaObject` y `DefinitionScope` para el primer caso, y la modificación de los metadatos correspondientes a `Module`, `Interface` y `Class` para el segundo.

5.2.3 Limitación para contemplar elementos temporales

Como se vio en el capítulo 3, los modelos temporales propuestos extendían ODMG para ampliarlo con nuevos tipos y elementos con semántica temporal. El hecho de que algunos de dichos modelos propongan una extensión de ODMG ya implica la propia limitación

del mismo para representar elementos temporales. También existe esta limitación en los metadatos de ODMG y por tanto es necesaria su ampliación para poder manejar dichos objetos temporales. Es necesario realizar ciertas modificaciones sobre los metadatos, de forma que se puedan saber qué objetos son temporales y cuales no lo son.

Se podría pensar que sería suficiente con realizar la ampliación siguiendo el metaesquema de la figura 5.2. En este caso sería necesario incluir una nueva *clase temporal* heredando de `class` y una nueva *interface temporal* heredando de `interface`. Incluso se podría incluir un *elemento temporal* entre `type` e `interface` de tal forma que a partir de ese objeto todos fueran temporales. Sin embargo, hay que tener en cuenta que necesitamos saber qué instancias de `MetaObject` están incluidas en un ámbito determinado, así como cuáles son las instancias de `Interface` y de `Class` junto con sus relaciones en una instancia de `Module`. Además, partimos de la extensión propuesta en trabajos anteriores [Torr02a], siendo necesaria la incorporación de los elementos temporales a partir de los metadatos ya existentes para así poder definir esquemas de exportación expresados en ODMG+T.

Esto supone la modificación de los metadatos correspondientes a `interface`, `derivedinterface`, `class`, `derivedclass` y `attribute`.

Por otra parte es necesario incluir información de las fuentes de datos en los metadatos de ODMG, para de esta forma saber que fuentes forman parte de un almacén y a que almacén (o almacenes) pertenece una determinada fuente. También en este caso es necesario modificar los metadatos, en concreto será necesario la modificación de los metadatos correspondientes a `Module` y la introducción de un nuevo elemento para representar las fuentes de datos.

5.3 Extensión de los metadatos de ODMG

Como se vio en el Capítulo 2, a partir del esquema componente (esquema conceptual enriquecido semánticamente) de una fuente de datos se definen los esquemas de exportación que ocultan parte de la información que por diversas razones (por que no se considere necesaria, por cuestiones de seguridad, etc.) no es de utilidad para el almacén de datos. Por tanto, el esquema de exportación se puede considerar con un esquema externo generado a partir de un esquema conceptual (esquema componente). Por otra parte, primero las fuentes de datos (junto con los métodos de extracción utilizados en cada una de ellas) se enriquecen con propiedades temporales y segundo, dada la naturaleza intrínsecamente temporal del almacén también es necesario disponer de objetos temporales (y no temporales) que representen a los elementos que forman parte del almacén.

Es necesario por tanto que en el metaesquema existan los conceptos de esquema externo y de clase derivada, así como las relaciones que representen los distintos objetos incluidos en los esquemas (clases, interfaces y demás). Además, debido a que algunas clases de los esquemas pueden estar definidas a partir de interfaces, si definimos clases derivadas a

partir de dichas clases ocultando algunas características de la interfaz, podría ser necesario generar interfaces derivadas que ocultasen dichas características comunes[Torr01a].

También tenemos que ampliar los metadatos para que puedan contemplar clases e interfaces temporales (derivadas o no) y atributos temporales. Cada almacén se alimenta de una o más fuentes de datos cuyos metadatos (utilizados en los algoritmos de integración y en el refresco) deben estar en el metaesquema, siendo también en este caso necesaria la ampliación de los metadatos de ODMG. En esta sección presentaremos cuáles son los metadatos propuestos para solventar los problemas antes mencionados

5.3.1 Metadatos para clases derivadas e interfaces derivadas

Un esquema de exportación (esquema externo) está formado por interfaces, clases y excepciones. Las interfaces y las clases pueden ser derivadas o no. En el mecanismo de definición propuesto en [Torr02a], las clases derivadas son clases de pleno derecho, sin pérdida de generalidad podemos decir que un módulo ODMG incluye clases e interfaces, las cuales pueden ser derivadas o no derivadas. Se trata pues de una abstracción de los conceptos de clase e interfaz que denominamos *clase genérica* e *interfaz genérica*. La figuras 5.3a y 5.3b ilustran esta abstracción y muestran la *relación de derivación*. En el caso de las clases, la relación de derivación expresa la relación existente entre las clases derivadas y sus clases base, que pueden ser derivadas o no derivadas, es decir, clases genéricas. De esta forma queda representado el hecho de que una clase derivada puede estar definida a partir de varias clases existentes, y que a partir de una clase existente se pueden definir varias clases derivadas. Análogamente, podemos realizar este mismo planteamiento para las interfaces derivadas, tal y como ilustra la Figura 5.3b.

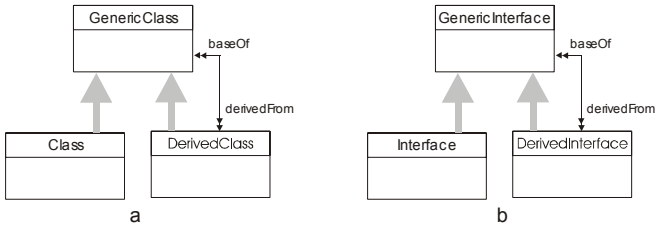

Figura 35. a) Metadatos para clases derivadas; b) Metadatos para interfaces derivadas

5.3.2 Metadatos para los componentes de un esquema

Es necesario modificar la definición de `Module` para que se puedan conocer todas las clases e interfaces incluidas en un módulo junto con sus respectivas relaciones de herencia en dicho módulo. Hasta ahora sólo es posible conocer en qué módulo se ha realizado la definición de una clase o una interfaz, pero no en qué módulo se utiliza. Conceptualmente, se trata de dos relaciones ternarias, una para interfaces y otra para clases. En el caso de la relación ternaria para las interfaces, tenemos interfaces que actúan

como superinterfaces, interfaces que actúan como subinterfaces y los módulos en los que existe esta relación. De esta forma, se puede conocer cuál es la relación de herencia existente entre dos interfaces por esquema. Asimismo, esta relación tiene que establecerse igualmente entre los módulos y las clases, dando lugar a la otra relación ternaria que permite representar cuál es la relación de herencia existente entre dos clases dentro de un esquema.

Es necesario incluir una nueva relación M:N entre `MetaObject` y `DefiningScope` para modelar la relación de utilización. Con esta nueva relación se podrá saber cuáles son los metaobjetos incluidos o utilizados en un ámbito concreto, así como en qué ámbitos se utiliza un metaobjeto determinado, ya que con la relación actual `definedIn` sólo se puede saber dónde se definió pero no dónde se utiliza.

La Figura 5.4 ilustra las modificaciones propuestas sobre algunos de los metadatos para modelar en el repositorio de esquemas de ODMG las interfaces y clases de un módulo, así como las relaciones de herencia por módulo.

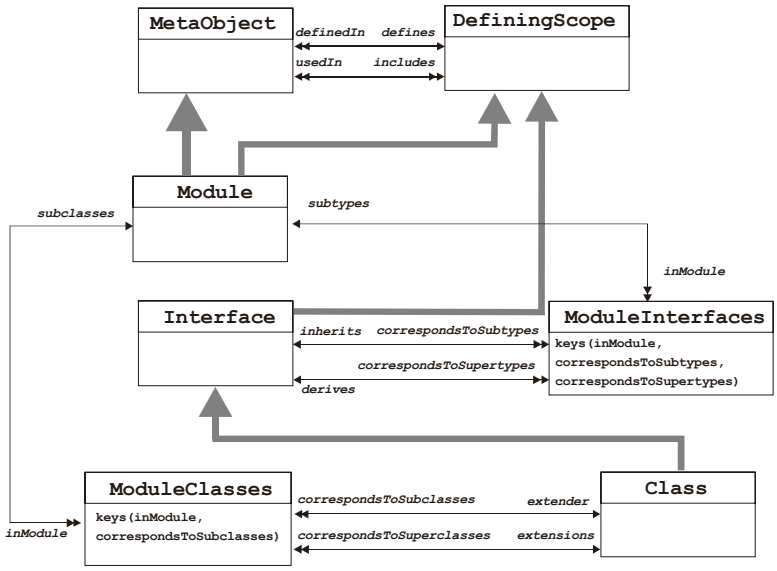

Figura 36. Representación de componentes de un esquema y sus relaciones por esquema

Un módulo se puede corresponder con varias instancias de la metaclase `ModuleInterfaces`, lo que está expresado por la relación `subtypes-inModule` que existe entre `Module` y `ModuleInterfaces`. Además, `ModuleInterfaces` está relacionada con `Interface` para determinar la interfaz a la que hacen referencia las instancias de dicha metaclase, y representar finalmente la relación ternaria. Estas relaciones son las dos relaciones que existen entre `ModuleInterfaces` e `Interface`.

Análogamente, se puede observar que se ha creado otra metaclase, denominada `ModuleClasses`, para expresar la relación ternaria existente entre las clases y los módulos. No obstante, como las clases sólo pueden tener relaciones de herencia simple, en cada

módulo sólo va a existir una superclase por clase, por lo que la clave se reduce a la clase que actúa como subclase y al módulo en el que existe esta relación.

5.3.3 Metadatos para las fuentes de datos del almacén

ODMG no incorpora metaobjetos específicos para la definición de fuentes de datos. La incorporación de este tipo de objetos implica la modificación de la definición de `Module` para que se puedan conocer todas fuentes que forman parte de un determinado almacén de datos, es decir, todas las fuentes incluidas en un módulo junto con sus respectivas características. Hasta ahora sólo es posible conocer en qué módulo se ha realizado la definición de un metaobjeto, pero no en qué módulo se utiliza. Como comentamos previamente una fuente puede formar parte de varios almacenes de datos y un almacén de datos se alimenta de más de una fuente de datos. El metaobjeto que se va a añadir a los metadatos lo denominamos `DataSource`.

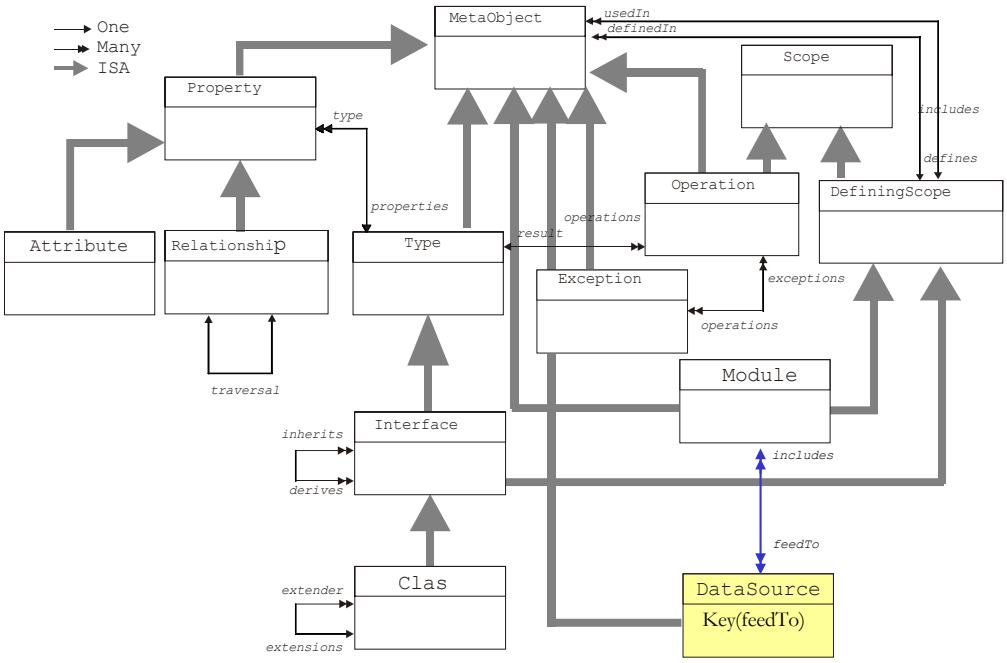

Figura 37. Fragmento de la estructura de metaobjetos de ODMG relacionado con las fuentes de datos

Será necesario incluir una relación entre `DataSource` y `Module` para saber que `DataSource` pertenecen a cada módulo y de que `DataSource` se compone cada módulo. Será una relación de muchos a muchos. La figura 5.5 ilustra esta abstracción y muestran la *relación de almacén-fuentes*, denominada `feedTo-includes`. De esta forma queda representado el hecho de que una fuente de datos puede alimentar varios almacenes, y que de un almacén pueden formar parte varias fuentes de datos.

En la figura 5.5 también se puede apreciar la relación entre `MetaObject` y `DefiningScope` que ya sé comento en el apartado anterior. De esta forma podremos saber cuáles son los metaobjetos incluidos o utilizados en un ámbito concreto, así como en qué ámbitos se utiliza un metaobjeto determinado.

5.3.4 Metadatos para metaclases temporales

Los modelos temporales distinguen entre elementos temporales y no temporales. Es decir, consideran importante que en un determinado esquema pueda haber definidos objetos con semántica temporal y otros que no tengan dicha cualidad. Parece por tanto necesario distinguir entre ambos tipos de objetos. Proponemos a continuación dos posibles opciones a la hora de ampliar los metadatos para incorporar elementos temporales.

En la primera opción, figura 5.6, los objetos temporales heredan de los no temporales, manteniendo de esta forma la posibilidad de definir objetos temporales y no temporales. De esta forma, cuando tengamos esquemas componentes podrá haber objetos de ambas clases, y al generar esquemas externos también se podrá tener objetos temporales o no temporales. Lo que se añade a los metadatos es lo siguiente: `attribute_T`, que hereda del objeto original de ODMG `attribute`; `class_T` que hereda de `class`; `derivedclass_T` que hereda de `derivedclass`; `interface_T` que hereda de `interface` y `derivedinterface_T` que hereda de `derivedinterface`.

En la segunda opción, figura 5.7, los objetos temporales heredan de `genericinterface`, de esta forma todos los objetos tendrían semántica temporal. La ampliación de los metadatos sería más sencilla, en este caso habría que añadir: `attribute_T`, que hereda del objeto original de ODMG `attribute`; `class_T` e `interface_T` que heredan de `genericinterface`.

Teniendo en cuenta lo que hemos comentado al principio de este apartado hemos elegido la opción 1, para así disponer de la posibilidad de definir objetos temporales y no temporales. La modificación del metaesquema con la inclusión de metaclases para clases e interfaces (derivadas o no) temporales y atributos temporales, no hacen que la estructura del repositorio de esquemas de ODMG deje de ser un esquema orientado a objetos.

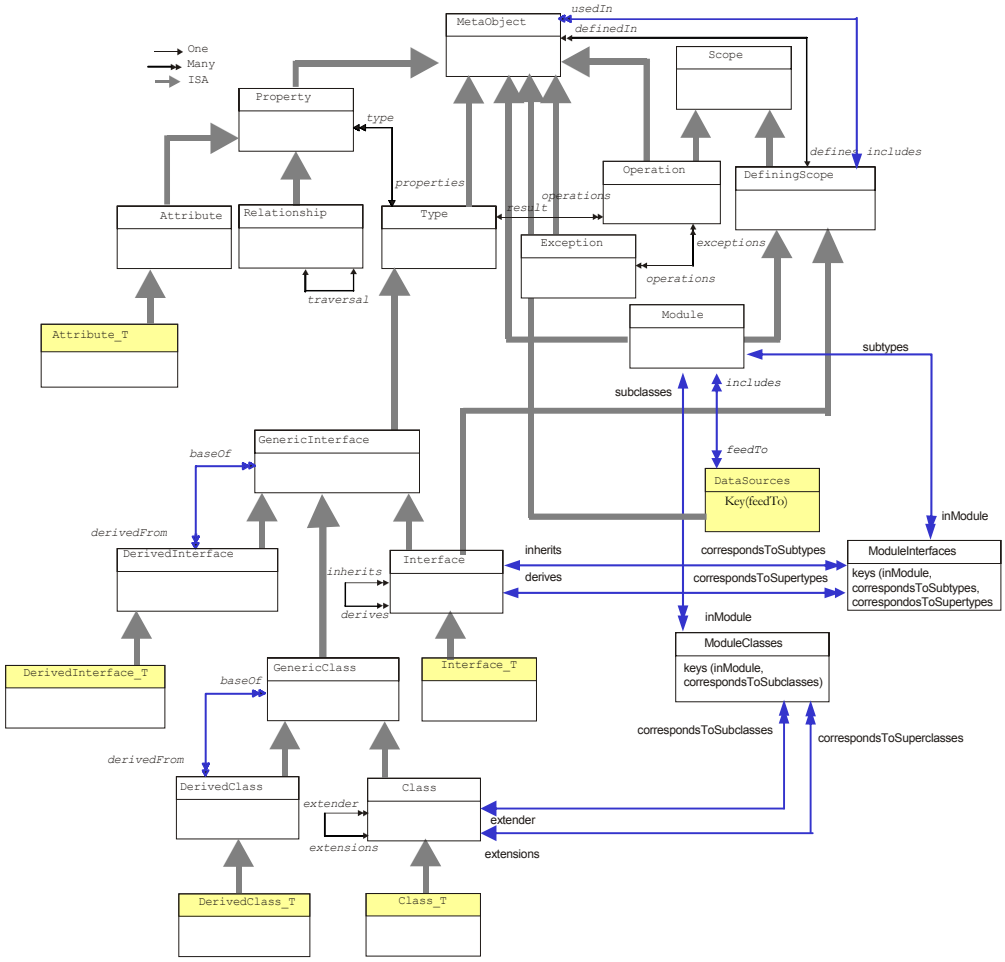

Figura 38. Opción 1

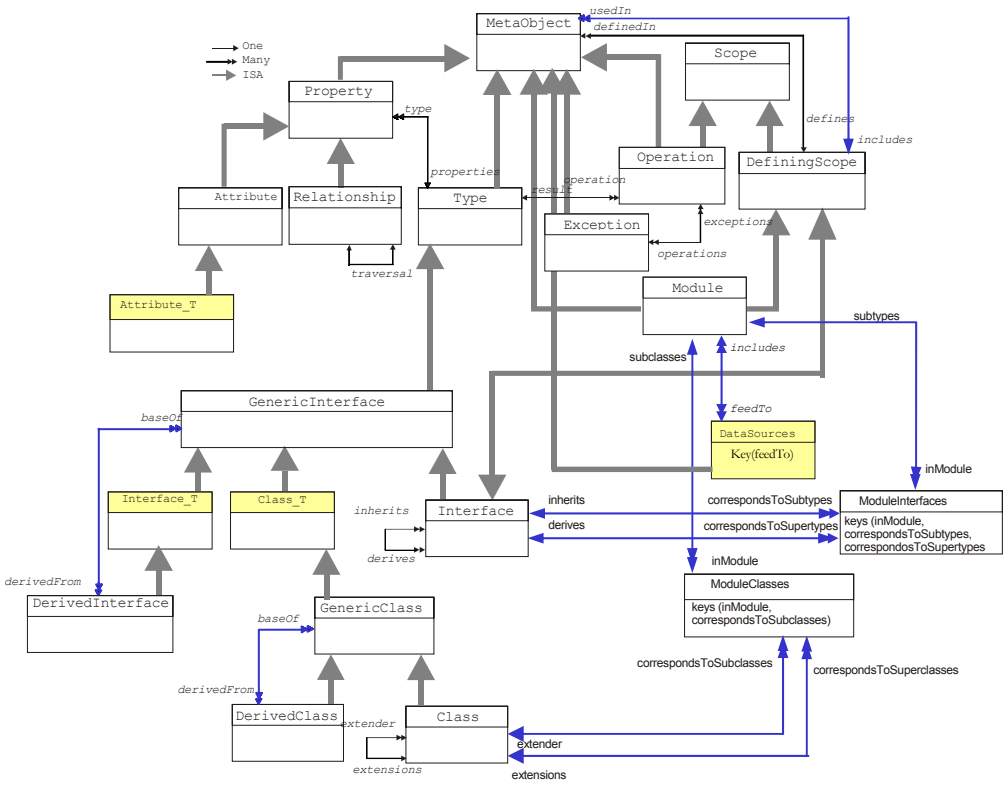

Figura 39. Opción 2

5.4 Definición en ODL de los metadatos propuestos

Una vez presentados los distintos metadatos propuestos, en esta sección se presentan las definiciones correspondientes en ODL, indicando lo que son las nuevas definiciones, así como las modificaciones realizadas sobre las definiciones existentes. Dichas modificaciones se indican en *cursiva* y sólo se incluye la especificación de los atributos y de las relaciones, dejando las operaciones y excepciones como puntos suspensivos. La definición de los metadatos se lleva a cabo como en el estándar ODMG 3.0, consistiendo en una definición interface para cada metaclase.

5.4.1 Interfaces

En ODMG 3.0 `Interface` se define a partir de `Type`. En cambio, la modificación propuesta en este capítulo incluye un elemento nuevo motivado por la introducción del concepto de interfaz temporal e interface temporal derivada. A continuación se incluye la definición existente en ODMG.

```
interface Interface: Type, DefiningScope {
    struct ParameterSpec {
        string param_name;
        Direction param_mode;
        Type param_type;};
    relationship set<Interface> inherits
        inverse Interface::derives;
    relationship set<Interface> derives
        inverse Interface::derives;
};
```

A continuación, se muestra la definición en ODL propuesta para los metadatos `Interface_T` y `DerivedInterface_T`.

```
interface Interface_T: Interface {
    ...
};

interface DerivedInterface_T: DerivedInterface {
    ...
};
```

5.4.2 Clases

Al igual que ocurre con la definición de los metadatos relacionados con las interfaces, se incluye un nuevo elemento motivado por la introducción del concepto de clase temporal y clase temporal derivada. A continuación se incluye la definición existente en ODMG.

```
interface Class:Interface {
    attribute list<string> extents;
    attribute list<string> keys;
    relationship Class extender
        inverse Class::extensions;
    relationship set<Class> extensions
        inverse Class::extender;
};
```

A continuación, se muestra la definición en ODL propuesta para los metadatos `Class_T` y `DerivedClass_T`.

```
interface Class_T: Class {
    ...
};

interface DerivedClass_T: DerivedClass {
    ...
};
```

5.4.3 Atributos

Para el caso de los metadatos relacionados con los atributos se incluye un nuevo elemento motivado por la introducción del concepto de atributo temporal. A continuación se incluye la definición existente en ODMG.

```
interface Attribute:Property {
    Attribute boolean is_read_only;
};
```

A continuación, se muestra la definición en ODL propuesta para los metadatos `Attribute_T`.

```
interface Attribute_T:Attribue {
    ...
};
```

5.4.4 Componentes de un módulo

Para terminar con la definición en ODL de los metadatos propuestos en este capítulo presentaremos la modificación propuesta para `Module`.

5.4.4.1 DataSource

Se muestra a continuación la definición propuesta para `DataSource`, que se especifica mediante `class` y no mediante `interface`. Esto se debe a que en ODMG, para que un tipo tenga una clave tiene que tener una extensión, y esto sólo es posible en una especificación `class`.

```
class DataSource
    ( extent TheDataSources) {
    atributte interval VentanaDisponibilidad;
    atributte interval TiempoAlmacenamiento;
    atributte interval TiempoExtraccion;
    atributte timestamp M;
    relationship set<Module> feedTo
        inverse Module::includes;
    keys feedTo
...
};
```

5.4.4.2 Componentes de un módulo

Para terminar con la definición en ODL de los metadatos propuestos en este capítulo presentaremos la modificación propuesta para `Module`, y las definiciones de los metadatos correspondientes a `DataSource`. A continuación, se muestra la extensión propuesta para `Module`, que permite conocer las fuentes que pertenecen a un módulo.

```
interface Module: MetaObject, DefiningScope {
    relationship set<ModuleInterfaces> subtypes
        inverse ModuleInterfaces::inModule;
    relationship set<ModuleClasses> subclasses
        inverse ModuleClasses::inModule;
    relationship set<DataSource> includes
        inverse DataSource::feedTo;
...
};
```

5.5 Ejemplo

A continuación se presenta un ejemplo para ilustrar el efecto que tendría la definición de un esquema de exportación a partir de un esquema componente en los metadatos a nivel de ocurrencia. También se añade al ejemplo los metadatos correspondientes a `DataSource`.

5.5.1 Esquemas y metadatos

La Figura 5.8 muestra el esquema conceptual que vamos a utilizar en este capítulo para ilustrar la definición de esquemas externos con el lenguaje propuesto. Se trata del esquema utilizado en otros capítulos de esta tesis para una de las fuentes de datos de la aplicación desarrollada para ayudar a la toma de decisiones de pilotos de vuelo libre.

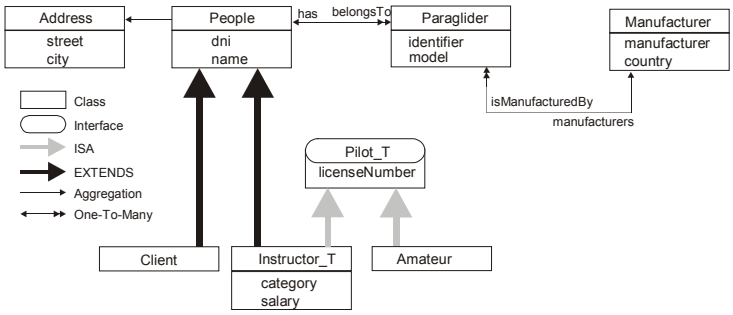

Figura 40. Esquema conceptual

La definición de este esquema crearía una serie de instancias en las metaclases `Class`, `Interface`, `Class_T`, `Interface_T`, `DataSource` y `Module` correspondientes a las clases e interfaces del esquema, así como a la definición del esquema en sí. También se crearían instancias en `ModuleClasses` y `ModuleInterfaces` para representar las relaciones `EXTENDS` e `ISA` existentes en el esquema conceptual, y se crearían las instancias de `DefiningScope` para el esquema, las clases y la interfaz, ya que son ámbitos de definición. Además, se crearían las instancias correspondientes a las propiedades y operaciones de las clases e interfaces. La Figura 5.10 ilustra las instancias de las metaclases `Module`, `Class`, `Class_T`, `Interface`, `DefiningScope`, `ModuleClasses` y `ModuleInterfaces` para el esquema de la Figura 5.8.

La metaclase Class tiene ocho instancias correspondientes a las siete clases del esquema más la clase Object. La metaclase `Interface_T` tendría la instancia correspondiente a la interface temporal `Pilot_T`, `Datasource` contiene las relaciones entre las diferentes fuentes de datos y los módulos definidos en el esquema y `Module` contendría la instancia

correspondiente al esquema conceptual. La metaclase `ModuleClasses`[3] contendría las relaciones `EXTENDS` de cada una de las clases del esquema, indicando para cada clase cuál es su superclase en un esquema determinado (p.e. la penúltima fila indicaría que la clase #306, `Confirmed`, es subclase de la clase #302, `People`, en el esquema #100, `ConceptualSchema`. Asimismo, la metaclase `ModuleInterfaces` contendría las relaciones `ISA` existentes en el esquema, indicando para cada clase o interfaz, las interfaces a partir de las que ha sido definida. La metaclase `DataSource` relaciona cada uno de los módulos con las fuentes de datos en los que se encuentran definidos. Por último, en `DefiningScope` se guarda la lista de componentes definidos y utilizados en cada ámbito de definición. En la figura 5.10 se observa como el ámbito de definición `ConceptualSchema` incluye las seis clases y la interfaz que forman dicho esquema.

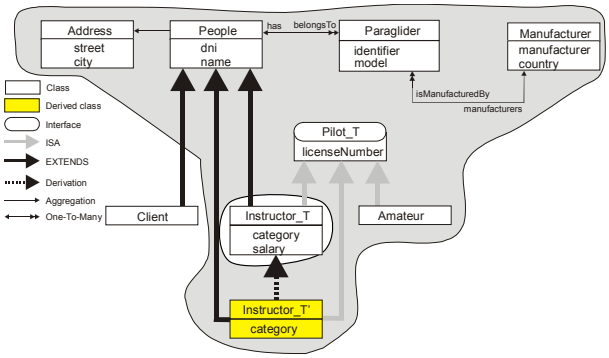

Figura 41. Esquema externo sin datos de clientes

Para ilustrar el funcionamiento de este lenguaje utilizaremos como ejemplo un esquema externo que oculta todos los datos de los clientes y el salario de los instructores, cuyo esquema resultante se ilustra en la figura 5.9. No obstante, la selección inicial no incluiría ninguna de las relaciones representadas en el esquema de la figura. Dichas relaciones son obtenidas posteriormente por los procesos de clausura y de generación de esquemas externos.

[3] Se ha elegido este diseño para `ModuleClasses` y `ModuleInterfaces` porque guarda explícitamente una instancia para cada relación de herencia. No obstante, se podría haber guardado una sola instancia por componente, guardando todas las componentes con las que está relacionada mediante herencia utilizando una lista. Sin embargo, el diseño propuesto permite acceder fácilmente a los componentes tanto por subclase como por superclase (interfaz y superinterfaces para el caso de interfaces).

Class

OID	name
#300	Object
#301	Address
#302	People
#303	Glider
#304	Manufacturer
#305	Client
#307	Student

Class_T

OID	name
#401	Instructor

Interface_T

OID	name
#200	Object
#201	Piloto

DataSource

OID	name	module
#700	WebApplication	#100

Module

OID	name	source
#100	ConceptualSchema	#700

ModuleClasses

OID	schema	class	superclass
#500	#100	#301	#300
#501	#100	#302	#300
#502	#100	#303	#300
#503	#100	#304	#300
#504	#100	#305	#302
#505	#100	#401	#302
#506	#100	#307	#300

ModuleInterfaces

OID	schema	component	superinterface
#600	#100	#201	#200
#601	#100	#401	#201
#602	#100	#307	#201

DefiningScope

OID	name	defines	includes
#100	ConceptualSchema	#201, #301, #302, #303, #304, #305, #401, #307	#201, #301, #302, #303, #304, #305, #401, #307
#201	Pilot	-	-
#300	Object	-	-
#301	Address	-	-
#302	People	-	-
#303	Glider	-	-
#304	Manufacturer	-	-
#305	Client	-	-

Figura 42. Metadatos correspondientes al esquema conceptual

5.5.2 Efecto de la definición de un esquema de exportación en el repositorio

En esta sección se ilustra cómo quedaría el repositorio una vez definido y generado el esquema externo de la figura 5.9. La definición de esquemas hace referencia al hecho de asignar componentes a los esquemas. El efecto de la definición de esquemas en el repositorio se traduce en la creación de una instancia para el esquema externo que se está definiendo, y la creación de un nuevo ámbito de definición al que se le añaden las clases e interfaces que componen dicho esquema a través de la relación `usedIn-defines`. Además, los procesos de clausura y de generación actualizan la información del repositorio tal y como se verá cuando se estudien en capítulos posteriores. Concretamente, si la clausura y la generación añaden nuevos componentes que son necesarios para el cierre y la corrección del esquema, se tendrán que actualizar los metadatos correspondientes a dichos componentes y se debe actualizar la relación `usedIn-defines`. Además, las relaciones de herencia descubiertas en el proceso de generación serán añadidas como instancias a `ModuleClasses` y `ModuleInterfaces`, en función de si son relaciones `EXTENDS` o `ISA`.

Class

OID	name
#300	Object
#301	Address
#302	People
#303	Glider
#304	Manufacturer
#305	Client
#307	Student

Class_T

OID	name
#401	Instructor
#402	**Instructor1**

Interface_T

OID	name
#200	Object
#201	Piloto

DataSource

OID	name	module
#700	WebApplication	#100
#700	**WebApplication**	**#101**

Module

OID	name	source
#100	ConceptualSchema	#700
#101	**DerivedSchema**	**#700**

ModuleClasses

OID	schema	class	superclass
#500	#100	#301	#300
#501	#100	#302	#300
#502	#100	#303	#300
#503	#100	#304	#300
#504	#100	#305	#302
#505	#100	#401	#302
#506	#100	#307	#300
#507	**#101**	**#301**	**#300**
#508	**#101**	**#302**	**#300**
#509	**#101**	**#303**	**#300**
#510	**#101**	**#304**	**#300**
#511	**#101**	**#307**	**#300**
#512	**#101**	**#402**	**#401**

ModuleInterfaces

OID	schema	component	superinterface
#600	#100	#201	#200
#601	#100	#401	#201
#602	#100	#307	#201

DefiningScope

OID	name	defines	includes
#100	ConceptualSc	#201, #301, #302, #303, #304, #305, #401, #307	#201, #301, #302, #303, #304, #305, #401, #307
#201	Pilot	-	-
#300	Object	-	-
#301	Address	-	-
#302	People	-	-
#303	Glider	-	-
#304	Manufacturer	-	-
#305	Client	-	-
#101	**DerivedSche**	**#402**	**#201, #301, #302, #303, #304, #402, #307**

Figura 43. El repositorio una vez definido el esquema externo

La Figura 5.11 ilustra cómo quedaría el repositorio una vez definido y generado el esquema externo de la Figura 5.8. En dicha figura, los nuevos metadatos aparecen en negrita. Concretamente, en `Module` hay una nueva instancia, correspondiente al esquema externo `derivedEsquema` que se ha creado. Como dicho esquema define un ámbito, también aparece una instancia para dicho esquema en `DefiningScope`, que además contiene la lista de componentes del esquema, es decir, las siete clases que forman el esquema `derivedEsquema`. Por último, las instancias de `ModuleClasses` corresponden a las relaciones `EXTENDS` que existirían entre las clases del esquema example. Como se puede observar en la figura, todas estas instancias menos una son extensiones a `Object`, por lo que se pueden ignorar. Se ha añadido un nuevo registro a la metaclase `DataSource` para indicar que el nuevo módulo derivado sigue perteneciendo a la misma fuente de datos.

5.6 Conclusiones

En este capítulo se ha propuesto la extensión de los metadatos de ODMG introduciendo las definiciones correspondientes a los metadatos para representar las fuentes de datos de las que se alimenta el almacén así como información temporal de las mismas. Así mismo

se han introducido los metadatos necesarios para saber que fuentes de datos alimentan a un almacén, o si una determinada fuente puede alimentar a más de uno. También se han extendido los metadatos para poder definir este tipo de información. Por otra parte también ha sido necesario extender los metadatos de clase, interface y atributo tal y como se definen en ODMG con otros que permitan contemplar conceptos temporales dando lugar a las clase temporal, interface temporal y atributo temporal. Como a partir de un esquema componente se pueden generar, si se considera necesario, esquemas de exportación, se ha realizado la extensión de los metadatos con clases e interfaces derivadas temporales.

Los metadatos propuestos se han presentado de forma gráfica, ilustrando la estructura propuesta del metaesquema, y en ODL, que es como aparecen definidos en el estándar ODMG.

Capítulo 6

ARQUITECTURA DE INTEGRACIÓN Y ALGORITMOS DE INTEGRACIÓN DE PROPIEDADES TEMPORALES DE LOS DATOS

La integración de información se ha estudiado desde diferentes áreas. Se han propuesto numerosas arquitecturas y propuestas para integrar datos de diferentes fuentes de datos heterogéneas, con diferentes modelos y esquemas de datos. Sin embargo, ninguna de las propuestas ha tenido en cuenta para la integración las propiedades temporales de los datos, y, para el caso del almacén de datos, tampoco se han estudiado los parámetros temporales de los métodos de extracción necesarios para obtener los datos de las fuentes. Por otra parte las arquitecturas propuestas tampoco han tenido en cuenta los aspectos temporales de los datos y se han centrado en la integración sintáctica y semántica de los datos. Por tanto es necesario disponer de algoritmos para la integración de las propiedades temporales de los datos que nos resuelvan si es posible integrar datos provenientes de las fuentes basándose en dichas propiedades.

El capítulo se estructura como sigue. En la sección 6.1 se introduce el problema y se definen los conceptos temporales necesarios. En la sección 6.2 se definen los parámetros relacionados con métodos de extracción. En la sección 6.3 se explica como se puede realizar la integración de los métodos de extracción según sus características temporales. En la sección 6.4 se propone una arquitectura de integración y los algoritmos para realizar la integración de las propiedades temporales de los datos. En la sección 6.5 mostramos un ejemplo de aplicación. Y finalizamos con las conclusiones en la sección 6.6.

6.1 Introducción

La posibilidad de integrar información de diferentes fuentes de datos es un campo de investigación muy estudiado. En [Hall00] se propone una arquitectura de tres niveles para la integración de fuentes de datos sobre información turística. En [Mour04] se presenta un sistema de ayuda a la decisión para operaciones de control en el espacio y en tiempo

real. En [Oliv01] se propone una metodología para la integración de políticas de seguridad multinivel en bases de datos federadas. En [Born99] se propone el uso de los metadatos para la integración de fuentes de datos.

En el ámbito de los almacenes de datos también se han estudiado y propuesto diferentes soluciones para la integración de información [Haas98] [Calv98] [Levy99] [Kurz98]. Los principales problemas a resolver en el contexto de los almacenes de datos son:

- *Heterogeneidad de las fuentes de datos*: podemos encontrar diferentes fuentes con diferentes formatos, como se explicó en el capítulo 2.
- *Evolución de las fuentes de datos*: cada fuente puede cambiar de estructura de forma autónoma. El administrador de la fuente de datos puede hacer los cambios que considere necesarios para su trabajo.
- *Autonomía de las fuentes de datos*: algunas fuentes tienen un alto grado de autonomía y su integración en el almacén suele ser de forma parcial y muy compleja.
- *Imposibilidad de encontrar metadatos de las fuentes*: en general, no es común encontrar metadatos disponibles sobre las características de la fuente de datos.

Sin embargo ninguna de las soluciones propuestas aborda el tema de la integración de las propiedades temporales de los datos. Por ejemplo si vamos a integrar dos datos diferentes, b y c, provenientes de diferentes fuentes, B y C, en uno solo en el almacén, que lo llamamos *a*, es posible que tengamos que tener en cuenta cuestiones temporales de las fuentes de datos que proporcionan los dato a integrar. Si la fuente B cambia de estado a un determinado ritmo y la fuente C a otro, hay que tener en cuenta dichos tiempos para saber en que momento realizar la integración de los datos b y c para transformarlos en *a*.

Es decir, la fase en la que se resuelven como se integran dos o más datos proveniente de diferentes fuentes (por ejemplo cambiando los formatos de los datos para ajustarlos al modelo de datos canónico) está resuelta por otros autores [Cast93]. Nosotros estamos en una fase en la que se decide si datos de diferentes fuentes puede ser integrado temporalmente. Por ejemplo, si de una fuente podemos obtener un dato con una granularidad de 24 horas (una vez al día) y de otra fuente otro dato (que se quiere integrar con el anterior en una fase posterior) con una granularidad de 1 hora, el resultado de la integración temporal es: podemos integrar temporalmente los dos datos cada 24 horas, cuando ambos estén disponibles.

6.1.1 Situación de partida

Partimos del modelo de tiempo propuesto en el capítulo 3. Antes de pasar a describir las características temporales de las fuentes de datos que van a ser integradas es necesario concretar la situación de partida. En nuestro caso se parte del componente de extracción del proceso ETL (Extracción, Transformación y Carga). Es decir, inicialmente, trataremos con tiempos asociados a la fuente de datos y a la parte de extracción de los mismos. Este paso previo se justifica sobre la base de que antes de realizar cualquier

transformación de los datos, es necesario saber si es posible, en cuanto a cuestiones temporales se refiere, llevar a cabo la integración de uno o más datos procedentes de una o más fuentes de datos.

Es decir, que estamos tratando tiempos en la parte de la fuente de datos y en el componente de extracción de los mismos. Esto es necesario antes de realizar la transformación de los datos para saber si es posible, en cuanto a cuestiones temporales se refiere, realizar la integración de uno o más datos provenientes de una o más fuentes de datos.

Transformar los datos (con cuestiones de cambio de formatos, etc.) y cargarlos en el DW conllevará otros tiempos que no se consideran en la fase de *integración de propiedades temporales* de los datos de las distintas fuentes de datos.

De igual forma, tampoco se van a tener en cuenta los tiempos empleados en resolver las heterogeneidades semánticas, puesto que no es el objetivo de este trabajo y, por otra parte, ya ha sido resuelto previamente en [Cast93]. Suponemos que vamos a integrar datos que previamente han pasado la fase de integración semántica.

6.1.2 Conceptos temporales

La figura 6.1 representa, sobre una línea de tiempo, los conceptos temporales a tener en la vida temporal de un dato, desde que cambia su valor en el mundo real (*VTinicio*) hasta que pasa a almacenamiento secundario por que no se considera de interés para análisis a corto plazo (*Z*). También se muestran los metadatos temporales de las fuente de datos (cada dato está en una determinada fuente con sus características temporales asociadas).

Están ordenados temporalmente, de forma que un concepto colocado a la izquierda de otro debe producirse antes que éste último para que la integración se realice de forma correcta. Algunos como *W*, puede ocurrir en diferentes momentos.

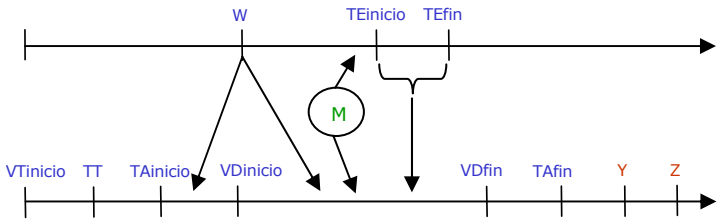

Figura 44. Vida temporal de un dato

Pasamos a continuación a definir dichos conceptos:

- *VTinicio (instant)*: instante de tiempo en el que un dato cambia en el mundo real. En este instante comienza su tiempo de validez (*VT*). El final del VT se puede aproximar de diferentes formas, las cuales dependerán del tipo de fuente y del método de extracción de los datos.

- *TT (instant)*: instante de tiempo en el que el dato se graba en el sistema informático de la fuente de datos. Se corresponderá con el tiempo de transacción.
- *VD (interval)*: Es la ventana de Disponibilidad, que se define como el intervalo de tiempo en el cual la fuente de datos puede ser accedida por parte de los programas de monitorización que se encargan de extraer los datos de las fuentes. Puede haber más de una ventana de disponibilidad diaria. Para cada VD:
  - *VDinicio*, instante de tiempo en el que se inicia la ventana de disponibilidad.
  - *VDfin*, instante de tiempo en el que finaliza la ventana de disponibilidad.
- *W(instant)*: instante de tiempo en el que el dato está disponible para ser consultado. Suponemos que puede pasar un intervalo de tiempo entre el instante en el que el dato está realmente almacenado en el sistema informático de la fuente de datos y el instante en el que el dato está disponible para ser consultado". A este respecto existen dos posibilidades:
  - Que *W*<*VDinicio*, en este caso el dato solamente está disponible a nivel local de la fuente o para determinados usuarios.
  - Que *VDinicio*<=*W*<*VDfin*, en este caso el dato estaría disponible para ser monitorizado por parte de los programas de extracción que se encargan de consultas las fuentes de datos.
- *M(instant)*: Instante de tiempo en el que se inicio el proceso de monitorización. Dependiendo del método de extracción de los datos puede coincidir con *TEinicio*.
- *TE(interval)*: Tiempo de Extracción. Periodo de tiempo que emplea el programa de monitorización/extracción en extraer los datos significativos de la fuente, donde:
  - *TEinicio*, instante de tiempo en el que se inicia la extracción de los datos.
  - *TEfin*, instante de tiempo en el que finaliza la extracción de los datos.
  - Suponemos que el TE está dentro de la VD por si fuera necesario consultar la fuente para extraer algún dato. Es decir, *VDinicio* < *TEinicio* < *TEfin* < *VDfin*.
- *TA(interval)*: Tiempo de Almacenamiento. Intervalo en el que un cambio está almacenado en la fuente de datos. Puede ocurrir que un fichero log o delta se almacene sólo por algún tiempo determinado. La integración debe llevarse a cabo, por tanto, antes de que finalice este intervalo. El inicio del intervalo coincide con el instante en el que el cambio es introducido en la fuente de datos. El fin del mismo debe ser posterior al instante en el que acaba la ventana de disponibilidad para que los datos hayan podido ser accedidos. Es decir, *VDfin* <= *TAfin*. Dónde:
  - *TAinicio*: instante en el que el cambio está almacenado en la fuente.
  - *TAfin*: instante en el que el cambio deja de estar almacenado en la fuente.

- *Y(instant)*: Instante de tiempo a partir del cual el dato está grabado en el almacén de datos.
- *Z(instant)*: Instante de tiempo a partir del cual un dato del almacén se sumariza, y pasa a otro tipo de almacenamiento por que no se consideran necesarios.

6.1.3 Intervalos de tiempo

Entre algunos de los conceptos temporales descritos anteriormente es posible definir un intervalo que va a influir de alguna manera concreta en el proceso de integración. Los intervalos a destacar se describen a continuación.

- *VTinicio – TT*: Periodo de ajuste del dato. Según [Inmo02], es aconsejable dejar pasar cierto tiempo desde que el dato cambia en el mundo real hasta que se graba en el sistema informático de la fuente de datos, por si dicho dato sufriera algún cambio. Si así sucediera, el cambio en el dato se haría en el sistema de la fuente y no habría que hacer el cambio también en el almacén de datos.
- *TT – W*: Periodo mínimo de tiempo a partir del cual el dato puede ser consultado en la fuente.
- *VDinicio – VDfin*: Ventana de disponibilidad. Definida por el administrador de la fuente de datos.
- *TEinicio – TEfin*: Periodo de tiempo que se tarda en extraer un dato de la fuente, incluyendo la parte de recorrer el fichero delta, o el log, etc.
- *TAinicio – TAfin*: Periodo de tiempo durante el cual está almacenado el fichero delta, log o la imagen de una fuente en la fuente de datos.
- *Y – W*: Periodo de tiempo que transcurre desde que el dato está disponible en la fuente hasta que está disponible en el almacén.
- *Y – Z*: Periodo de tiempo que permanece un dato en el almacén hasta que pasa a otro tipo de almacenamiento.
- *M+t*: intervalo de tiempo que transcurren entre dos monitorizaciones de la fuente.

6.2 Parámetros temporales y métodos de extracción

Los parámetros relacionados con métodos de extracción (descritos en el capítulo 2) y mencionados en los apartados anteriores son descritos a continuación para una fuente de datos abstracta. Dependiendo del tipo de fuente concreta con la que se este tratando podremos especificar mejor algunos de estos parámetros.

6.2.1 Tiempo de ajuste

Es el momento en el que graba el dato en el Sistema Informático de la fuente de datos (*TA*). Es el TT de la fuente. A nivel de tipo de fuente no es posible especificar mejor este parámetro, ya que depende más de la fuente en concreto que del tipo de fuente.

Application Assisted	>0. Dependiente de la lógica de la aplicación
Timestamp Based	>0. Dependiente de la lógica de la aplicación
Triggered "Delta"	>0. Dependiente de la lógica de la aplicación
Triggered "Direct"	>0. Dependiente de la lógica de la aplicación
File Comparison	>0. Dependiente de la lógica de la aplicación
Log	>0. Dependiente de la lógica de la aplicación

6.2.2 Tiempo mínimo de espera para consulta

Es el tiempo que transcurre desde que el dato está en el Sistema Informático de la fuente hasta que está disponible para extraerlo de la fuente (*TMC*).

Application Assisted	>0. Tiempo que se tarda en introducir el cambio en el fichero delta.
Timestamp Based	>0. Una vez que se lanza la consulta que actualiza el valor modificado sobre la fuente de datos, es necesario esperar a que se considere "cometida".
Triggered "Delta"	>0. Se trata del intervalo que transcurre desde que se lanza la consulta que actualiza el valor modificado hasta que el trigger que se dispara automáticamente actualiza el fichero delta con este nuevo cambio.
Triggered "Direct"	0. El trigger se dispara automáticamente una vez que se produce el cambio y se encarga de enviar el cambio a la etapa de Transformación, por lo que en ningún caso será necesario consultar este tipo de fuente.
File Comparison	>0. Tiempo que se tarda en modificar el fichero.
Log	>0. Tiempo que se tarda en introducir el cambio en el fichero log.

6.2.3 Tiempo de extracción

Periodo de tiempo que emplea el programa de monitorización/extracción en extraer los datos significativos de la fuente (*TE*).

Application Assisted	>0. Tiempo que se emplea en recorrer todo el fichero delta, obtener los cambios recogidos en él. (y eliminarlo).
Timestamp Based	>0. Tiempo que se emplea en determinar los cambios que se han producido desde el último refresco.
Triggered "Delta"	>0. Tiempo que se emplea en determinar los cambios que se han producido desde el último refresco (recorrer todo el fichero delta y obtener los cambios recogidos en él).
Triggered "Direct"	0. El trigger se encarga de poner automáticamente y de forma prácticamente instantánea el valor que se ha modificado a disposición de la fase de transformación.

File Comparison	>>0. Tiempo empleado en recorrer la última imagen guardada del fichero y la versión actual, detectar los cambios producidos y actualizar la última imagen de la fuente de datos.
Log	>0. Tiempo empleado en recorrer el log y determinar los cambios producidos desde el último refresco.

6.2.4 Ventana de disponibilidad

Se define como el intervalo de tiempo en el cual la fuente de datos puede ser accedida por parte de los programas de monitorización que se encargan de extraer los datos de las fuentes (*VD*). Puede haber más de una ventana de disponibilidad diaria. El administrador del sistema informático de la fuente de datos es el encargado de definir la ventana de disponibilidad.

Application Assisted	>0. Intervalo de tiempo en el que es accesible el fichero delta.
Timestamp Based	>0. Intervalo de tiempo en el que es accesible la fuente de datos.
Triggered "Delta"	>0. Intervalo de tiempo en el que es accesible el fichero delta.
Triggered "Direct"	0. El trigger se dispara automáticamente una vez que se produce el cambio y se encarga enviar el valor que se ha modificado a disposición de la fase de transformación, por lo que no será necesario consultar este tipo de fuente.
File Comparison	>0. Intervalo de tiempo en el que es accesible el fichero.
Log	>0. Intervalo de tiempo en el que es accesible el log.

6.2.5 Periodo de muestreo

Instante de tiempo en el que se inicio el proceso de monitorización (*M*). Esta monitorización de la fuente puede ser periódica o no, dependiendo del tipo de fuente, del método de extracción asociado a la fuente y de los requerimientos del diseñador del almacén de datos.

Application Assisted	>0. Intervalo de tiempo que debe transcurrir entre procesamientos del fichero delta.
Timestamp Based	>0. Intervalo de tiempo que debe transcurrir entre consultas a de la fuente en busca de cambios producidos.
Triggered "Delta"	>0. Intervalo de tiempo que debe transcurrir entre procesamientos del fichero delta.
Triggered "Direct"	0. Los cambios producidos se envían a la etapa de Transformación en el momento en el que se producen.
File Comparison	>0. Intervalo de tiempo que debe transcurrir entre consultas a de la fuente en

	busca de cambios producidos.
Log	>0. Intervalo de tiempo que debe transcurrir entre consultas a de la fuente en busca de cambios producidos.

6.2.6 Tiempo de Almacenamiento del delta, log o imagen

Intervalo en el que un cambio está almacenado en la fuente de datos (*TALM*). Puede ocurrir que un fichero log o delta se almacene sólo por algún tiempo determinado, por ejemplo que el fichero log se borre al final de cada semana. La integración debe llevarse a cabo, por tanto, antes de que finalice este intervalo.

Application Assisted	>0. Intervalo máximo de tiempo que está almacenado el fichero delta hasta que se reinicializa.
Timestamp Based	0. NULL
Triggered "Delta"	>0. Intervalo máximo de tiempo que está almacenado el fichero delta hasta que se reinicializa.
Triggered "Direct"	0. NULL
File Comparison	>0. Intervalo máximo de tiempo que está almacenado el fichero imagen de la fuente.
Log	>0. Intervalo máximo de tiempo que está almacenado el fichero log hasta que se reinicializa.

6.2.7 Transaction Time

Instante de tiempo en el que el dato se graba en el sistema informático de la fuente de datos (*TT*). Se corresponderá con el tiempo de transacción. "Sí" indica que le método en cuestión proporciona este parámetro (bajo determinadas condiciones que se exponen a continuación) y "No" que en ningún caso lo puede proporcionar.

Application Assisted	Sí. Si la aplicación operacional se modifica o diseña de tal forma que marque temporalmente los cambios producidos.
Timestamp Based	Sí. El propio método de extracción se basa en el marcado temporal de los cambios producidos.
Triggered "Delta"	Sí. Si el fichero delta incluye un campo para almacenar una marca temporal.
Triggered "Direct"	Sí. Si en el diseño del trigger se incluye la funcionalidad necesaria para almacenar en el almacén el instante en el que se dispara.
File Comparison	No. No es posible determinar el instante en el que se producen los cambios. Se puede aproximar al momento en el que se realiza la comparación de las dos imágenes.

Log	Sí. Si el fichero delta incluye un campo para almacenar una marca temporal.

6.2.8 Disponibilidad de los cambios producidos

Dependiendo del método de extracción podremos disponer de todos o de algunos de los cambios producidos en la fuente de datos (*DCP*). Por ejemplo, en las fuentes con ficheros log o ficheros delta están registrados todos los cambios producidos en la fuente, solo hay que recorrerlos para averiguar los cambios en los datos desde la última consulta. Esto no ocurre por ejemplo con el método File Comparison dado que entre dos consultas a la fuente se pueden producir cambios en los datos que no quedan reflejados en ningún fichero.

Application Assisted	Todos los que nos interesen
Timestamp Based	Algunos
Triggered "Delta"	Todos los que nos interesen
Triggered "Direct"	Todos los que nos interesen
File Comparison	Algunos
Log	Todos

6.3 Integración de los métodos según sus características temporales

El problema de la integración de datos provenientes de las fuentes consiste básicamente en la integración temporal de un cambio producido en dos fuentes de datos concretas. Una vez concretadas las fuentes de datos y sus características podemos especificar con más detalle los requisitos y particularidades que existen a la hora de realizar la integración. Para cada uno de los cruces se realiza a continuación un estudio de los requisitos que deben cumplir cada uno de los parámetros descritos en el apartado 6.2 para llevar a cabo la integración.

		Método A (FD1)							
		TA	TMC	TE	VD	PM	TT	TAL	DC
Método B (FD2)	TA	1							
	TMC		2						
	TE			3					
	VD				4				
	PM					5			
	TT						6		
	TAL							7	
	DC								8

Tabla 5. Cruces de métodos según los parámetros temporales.

En la Tabla 5 las columnas y filas representan cada uno de los parámetros temporales y los números el resultado de dicha integración.

6.3.1 Método Application Assisted con el resto de métodos

6.3.1.1 Application Assisted – Application Assisted

En este caso realizamos la "integración temporal" de datos (provenientes de la misma o de diferentes fuentes) que son extraídos con el mismo método. Dado que con este método de extracción se pueden seleccionar los datos (los que estime oportunos el administrador del almacén) que el programa de aplicación modificado envía al fichero delta para su posterior procesamiento, podemos obtener todos los cambios de interés producidos en la fuente de datos.

A nivel abstracto podemos decir que se puede realizar la "integración temporal" a lo largo de todos los parámetros o características temporales antes mencionadas (ver 6.2).

El resultado de los cruces sería:

1. Será la aplicación operacional que necesite el mayor tiempo la que condicione la integración. Además el tiempo de ajuste debe acabar antes del inicio de la Ventana de Disponibilidad.
2. Será la aplicación operacional que necesite el mayor tiempo para escribir el cambio en el fichero delta la que condicione la integración.
3. Será la aplicación operacional que genere el fichero delta más grande la que necesite el mayor tiempo para extraer el cambio.

4. Se podrá realizar un refresco de los datos del DW cuando ambos ficheros delta sean accesibles.
5. El periodo de muestreo necesario para integrar las dos fuentes de datos deberá ser tal que coincida dentro de las ventanas de disponibilidad de ambas.
6. Si es el mismo dato: El TT de un cambio concreto recogido por ambas fuentes será el menor de las dos. Si son distintos datos también.
7. El único requisito es que el TAL esté incluido dentro de la Ventana de disponibilidad.
8. Se pueden obtener todos los cambios, puesto que ambas fuentes de datos pueden proveer de todos los cambios en los registros que nos interesan.

6.3.1.2 Application Assisted – Timestamp

De la fuente Application Assisted disponemos de todos los cambios de interés producidos. De la fuente con el método Timestamp solo obtendremos algunos de los cambios producidos (depende de la monitorización de la fuente). Por tanto podremos "integrar temporalmente" solo algunos de los cambios producidos en ambas fuentes. Respecto a la integración del parámetro "Tiempo de Almacenamiento" no sería posible de forma directa dado que el método Timestamp no dispone de dicho parámetro.

A nivel abstracto podemos decir que se puede realizar la "integración temporal" a lo largo de todos los parámetros o características temporales antes mencionadas (ver 6.2) excepto en el de DCP y TA.

Existe un caso particular que se da cuando los cambios en la fuente de datos monitorizada con el método de extracción Timestamp se producen de forma periódica, ya que ajustando el parámetro temporal "periodo de muestreo" a la periodicidad con que se producen los cambios en la fuente podríamos obtener todos los cambios de datos producidos en la misma.

El resultado de los cruces es:

1. En la integración de fuentes de datos con estos métodos de extracción influirá el tiempo de ajuste de aquella que tenga el mayor valor.
2. Hay que tener en cuenta el mayor de los dos tiempos.
3. El tiempo de extracción a tener en cuenta a la hora de integrar los datos será el que implique un mayor tiempo de procesamiento en ambas fuentes.
4. Se podrá realizar un refresco de los datos del DW cuando la fuente Timestamp y el fichero delta sean accesibles.
5. El periodo de muestreo necesario para integrar las dos fuentes de datos deberá ser tal que coincida dentro de las ventanas de disponibilidad de ambas.
6. Si es el mismo dato: El TT de un cambio concreto recogido por ambas fuentes será el menor de las dos. Si son distintos datos también.

7. El tiempo de almacenamiento de la FD1 debe estar incluido dentro del intervalo de monitorización de la FD2.
8. Con el método Application Assisted podemos tener todos los cambios de interés producidos, mientras que de la fuente con el método Timestamp solo obtendremos algunos de los cambios producidos (dependiendo de cómo se realice la monitorización de la fuente). Por tanto podremos "integrar temporalmente" solo algunos de los cambios producidos en ambas fuentes.

6.3.1.3 Application Assisted – Trigger Delta

De ambas fuentes tenemos todos los cambios de interés producidos. Por tanto podremos "integrar temporalmente" todos los cambios.

Podemos decir que se puede realizar la "integración temporal" de todos los parámetros temporales antes mencionadas (ver 6.2).

El resultado de los cruces es:

1. En la integración de fuentes de datos con estos métodos de extracción influirá el tiempo de ajuste de aquella que tenga el mayor valor.
2. En la integración de fuentes de datos con estos métodos de extracción influirá el tiempo de consulta de aquella que tenga el mayor valor.
3. Será la aplicación operacional que genere el fichero delta más grande la que necesite el mayor tiempo para extraer el cambio.
4. Se podrá realizar un refresco de los datos del DW cuando ambos ficheros delta sean accesibles.
5. El periodo de muestreo necesario para integrar las dos fuentes de datos deberá ser tal que coincida dentro de las ventanas de disponibilidad de ambas.
6. Si es el mismo dato: El TT de un cambio concreto recogido por ambas fuentes será el menor de las dos. Si son distintos datos también.
7. El tiempo de almacenamiento de las fuentes debe estar al menos incluido en la ventana de disponibilidad de ambas.
8. Se pueden obtener todos los cambios, puesto que de ambas fuentes de datos se pueden proporcionar todos los cambios en los registros que nos interesan.

6.3.1.4 Application Assisted – Trigger Directo

El método Application Assisted graba los movimientos de la fuente en un fichero delta del que podemos obtener todos los cambios. Con el método Trigger Directo también se pueden obtener todos los cambios dado que se dispara un trigger cada vez que se produce algún cambio en los datos (elegidos según las necesidades de la aplicación), sin embargo la propagación de los cambios es inmediata y no se almacenan en ningún fichero o etapa intermedia.

En general se puede deducir que se puede realizar la "integración temporal" a lo largo de todos los parámetros o características temporales antes mencionadas (ver 6.2), aunque dado que el método de extracción Trigger Directo no requiere consultar la fuente de datos(se dispara el trigger y se propaga el cambio) se podrían imponer algunas restricciones a la hora de realizar la "integración temporal".

El resultado de los cruces es:

1. Será la aplicación operacional que necesite el mayor tiempo para escribir el cambio la que condicione la integración.
2. Sólo influye el tiempo necesario en introducir el cambio en el fichero delta de la fuente Application Assisted, porque en el caso de la fuente Triggered direct este valor es nulo.
3. Sólo habrá que tener en cuenta el tiempo necesario para procesar el fichero delta de la fuente en la que se utiliza el método de extracción Application Assisted, pues en la otra fuente no se requiere tiempo en el proceso de extracción.
4. Se podrá realizar un refresco de los datos del DW cuando el fichero delta sea accesible.
5. El proceso de extracción de los datos de la fuente del tipo triggered direct deberá producirse dentro de la ventana de disponibilidad de la fuente del tipo Application Assisted para poder realizar la integración entre ambas.
6. Si es el mismo dato: El TT de un cambio concreto recogido por ambas fuentes será el menor de las dos. Si son distintos datos también.
7. No existe Tiempo de Almacenamiento en el Trigger Directo.
8. Se pueden obtener todos los cambios, puesto que ambas fuentes de datos pueden proveer de todos los cambios en los registros que nos interesan.

6.3.1.5 Application Assisted – File Comparison

Como hemos mencionado anteriormente con el método Application Assisted podemos tener todos los cambios producidos, mientras que con el método File Comparison solo obtendremos algunos de los cambios producidos (dependiendo de cómo se realice la monitorización de la fuente). Por tanto podremos "integrar temporalmente" solo algunos de los cambios producidos en ambas fuentes. Respecto a la integración del parámetro "TT" no sería posible dado que el método File Comparison no dispone de dicho parámetro.

Se puede realizar la "integración temporal" a lo largo de todos los parámetros temporales antes mencionadas (ver 6.2) excepto TT.

El resultado de los cruces es:

1. Será la aplicación operacional que necesite el mayor tiempo para escribir el cambio la que condicione la integración.

2. Influye la aplicación operacional que necesite más tiempo para modificar el fichero con los cambios.
3. Estará determinado por el tiempo necesario en realizar la comparación de las imágenes de la fuente basada en este método de extracción, ya que generalmente es un proceso más lento que la carga del fichero delta de la otra fuente.
4. Se podrá realizar un refresco de los datos del DW cuando ambos ficheros sean accesibles.
5. El periodo de muestreo necesario para integrar las dos fuentes de datos deberá ser tal que coincida dentro de las ventanas de disponibilidad de ambas.
6. No es posible integrar las fuentes de datos usando este valor porque la fuente del tipo File Comparison no provee de esta información.
7. El Tiempo de Almacenamiento debe estar incluido dentro de las Ventanas de Disponibilidad de ambas fuentes.
8. Con el método Application Assisted podemos tener todos los cambios de interés producidos, mientras que de la fuente con el método File Comparison sólo obtendremos algunos de los cambios producidos (dependiendo de cómo se realice la monitorización de la fuente). Por tanto podremos "integrar temporalmente" solo algunos de los cambios producidos en ambas fuentes.

6.3.1.6 Application Assisted – LOG

Ambos métodos producen un fichero con los cambios, sea un delta o un log, por tanto podremos "integrar temporalmente" todos los cambios producidos en las fuentes.

El resultado de los cruces es:

1. Será la aplicación operacional que necesite el mayor tiempo para escribir el cambio la que condicione la integración.
2. Influye la aplicación operacional que necesite más tiempo para modificar el fichero con los cambios.
3. El tiempo de extracción a tener en cuenta a la hora de integrar los datos será el que implique un mayor tiempo de procesamiento en ambas fuentes.
4. Se podrá realizar un refresco de los datos del almacén cuando ambos ficheros sean accesibles.
5. El periodo de muestreo necesario para integrar las dos fuentes de datos deberá ser tal que coincida dentro de las ventanas de disponibilidad de ambas.
6. Si es el mismo dato: El TT de un cambio concreto recogido por ambas fuentes será el menor de las dos. Si son distintos datos también.
7. El Tiempo de Almacenamiento debe estar incluido dentro de las Ventanas de Disponibilidad de ambas fuentes.

8. Se pueden obtener todos los cambios, puesto que ambas fuentes de datos pueden proveer de todos los cambios en los registros que nos interesan.

6.3.2 Método Timestamp con el resto de métodos

6.3.2.1 Timestamp – Timestamp

En este caso realizamos la "integración temporal" de datos (provenientes de la misma o de diferentes fuentes) que son extraídos con el mismo método. Con el método Timestamp solo obtendremos algunos de los cambios producidos (dependiendo de cómo se realice la monitorización de la fuente). Por tanto podremos "integrar temporalmente" algunos cambios.

El resultado de los cruces será:

1. Será la aplicación operacional que necesite el mayor tiempo para escribir el cambio la que condicione la integración.
2. La fuente de datos que necesite más tiempo para realizar los cambios será la que determine el tiempo mínimo de espera para realizar la consulta de estas.
3. El tiempo de extracción a tener en cuenta a la hora de integrar los datos será el que implique un mayor tiempo de procesamiento en ambas fuentes.
4. Se podrá realizar un refresco de los datos del almacén cuando ambas fuentes sean accesibles.
5. El periodo de muestreo necesario para integrar las dos fuentes de datos deberá ser tal que coincida dentro de las ventanas de disponibilidad de ambas.
6. Si es el mismo dato: El TT de un cambio concreto recogido por ambas fuentes será el menor de las dos. Si son distintos datos también.
7. No es necesario tenerlo en cuenta, pues en ambas fuentes de datos se almacenan los datos indefinidamente.
8. El número de cambios detectados por el conjunto de las fuentes será mayor o igual que los detectados por sólo una de ellas, pero nunca se puede asegurar que se detecten todos. Podremos integrar temporalmente todos aquellos cambios que aparezcan en ambas fuentes de datos.

6.3.2.2 Timestamp – Trigger Delta

Con el método Trigger Delta se graban en el fichero delta todos los cambios producidos, mientras que de la fuente con el método Timestamp solo algunos de los cambios (dependiendo de cómo se realice la monitorización de la fuente). Por tanto podremos "integrar temporalmente" solo algunos de los cambios. Respecto a la integración del parámetro "Tiempo de Almacenamiento" no sería posible de forma directa dado que el método Timestamp no dispone de dicho parámetro.

En general, se puede realizar la "integración temporal" a lo largo de todos los parámetros temporales (ver 6.2) excepto en el de DCP y TA.

En este caso el resultado de los cruces será:

1. Será la aplicación operacional que necesite el mayor tiempo para escribir el cambio la que condicione la integración.
2. La fuente de datos que necesite más tiempo para realizar los cambios será la que determine el tiempo mínimo de espera para realizar la consulta de estas.
3. El tiempo de extracción a tener en cuenta a la hora de integrar los datos será el que implique un mayor tiempo de procesamiento en ambas fuentes.
4. Se podrá realizar un refresco de los datos del almacén cuando ambas fuentes sean accesibles.
5. El periodo de muestreo necesario para integrar las dos fuentes de datos deberá ser tal que coincida dentro de las ventanas de disponibilidad de ambas.
6. Si es el mismo dato: El TT de un cambio concreto recogido por ambas fuentes será el menor de las dos. Si son distintos datos también.
7. El Tiempo de Almacenamiento del fichero delta debe estar incluido dentro de la ventana de disponibilidad de ambas fuentes.
8. Con el método Trigger Directo podemos tener todos los cambios de interés producidos, mientras que de la fuente con el método Timestamp sólo obtendremos algunos de los cambios producidos (dependiendo de cómo se realice la monitorización de la fuente). Por tanto podremos "integrar temporalmente" solo algunos de los cambios producidos en ambas fuentes.

6.3.2.3 Timestamp – Trigger Directo

Con Trigger Directo todos los cambios, mientras con el método Timestamp solo algunos de los cambios. Por tanto podremos "integrar temporalmente" solo algunos de los cambios producidos.

Se puede realizar la "integración temporal" a lo largo de todos los parámetros temporales excepto en el de DCP y TA (ver 6.2).

En este caso el resultado de los cruces será:

1. Será la aplicación operacional que necesite el mayor tiempo para escribir el cambio la que condicione la integración.
2. Sólo influirá el valor de la fuente de datos del tipo "timestamp-based" en la integración, pues en el caso de la otra fuente de datos no es necesario consultarla.
3. Sólo influirá el valor de la fuente de datos del tipo "timestamp-based" en la integración, pues en el caso de la otra fuente de datos no es necesario consultarla.

4. Se podrá realizar un refresco de los datos del almacén cuando la fuente de datos del tipo "timestamp-based" sea accesible, pues no es necesario consultar la otra fuente de datos.
5. El proceso de extracción de los datos de la fuente del tipo triggered direct deberá producirse dentro de la ventana de disponibilidad de la fuente del tipo Timestamp para poder realizar la integración entre ambas.
6. Si es el mismo dato: El TT de un cambio concreto recogido por ambas fuentes será el menor de las dos. Si son distintos datos también.
7. No es necesario tenerlo en cuenta, pues en ambas fuentes de datos se almacenan los datos indefinidamente.
8. Con el método Trigger Directo podemos tener todos los cambios de interés producidos, mientras que de la fuente con el método Timestamp sólo obtendremos algunos de los cambios producidos (dependiendo de cómo se realice la monitorización de la fuente). Por tanto podremos "integrar temporalmente" solo algunos de los cambios producidos en ambas fuentes.

6.3.2.4 Timestamp – File Comparison

Tanto con Timestamp como con File Comparison disponemos solo de algunos de los cambios producidos. Se pueden "integrar temporalmente" solo algunos de los cambios. En cuanto a la integración del parámetro "TT" no sería posible dado que el método File Comparison no dispone de dicho parámetro.

Se puede realizar la "integración temporal" de todos los parámetros excepto en el de DCP, TA y TT.

En este caso el resultado de los cruces será:

1. Será la aplicación operacional que necesite el mayor tiempo para escribir el cambio la que condicione la integración.
2. Influye la aplicación operacional que necesite más tiempo para realizar los cambios.
3. Estará determinado por el tiempo necesario en realizar la comparación de las imágenes de la fuente basada en este método de extracción, ya que generalmente es un proceso más lento que la consulta de la base de datos.
4. Se podrá realizar un refresco de los datos del almacén cuando ambas fuentes sean accesibles.
5. El periodo de muestreo necesario para integrar las dos fuentes de datos deberá ser tal que coincida dentro de las ventanas de disponibilidad de ambas.
6. No es posible integrar las fuentes de datos usando este valor porque la fuente del tipo "file comparison" no provee de esta información.
7. No es necesario tenerlo en cuenta, pues en ambas fuentes de datos se almacenan los datos indefinidamente.

8. El número de cambios detectados por el conjunto de las fuentes será mayor o igual que los detectados por sólo una de ellas, pero nunca se puede asegurar que se detecten todos. Podremos integrar temporalmente todos aquellos cambios que aparezcan en ambas fuentes de datos.

6.3.2.5 Timestamp – LOG

Con el método LOG tenemos todos los cambios y con el método Timestamp solo algunos. Se podrán "integrar temporalmente" solo algunos de los cambios.

La "integración temporal" se puede realizar a lo largo de todos los parámetros temporales de la sección 6.2 excepto en el de DCP y TA.

En este caso el resultado de los cruces es:

1. Será la aplicación operacional que necesite el mayor tiempo para escribir el cambio la que condicione la integración.
2. Influye la aplicación operacional que necesite más tiempo para realizar los cambios.
3. El tiempo de extracción a tener en cuenta a la hora de integrar los datos será el que implique un mayor tiempo de procesamiento en ambas fuentes.
4. Se podrá realizar un refresco de los datos del almacén cuando ambas fuentes sean accesibles.
5. El periodo de muestreo necesario para integrar las dos fuentes de datos deberá ser tal que coincida dentro de las ventanas de disponibilidad de ambas.
6. Si es el mismo dato: El TT de un cambio concreto recogido por ambas fuentes será el menor de las dos. Si son distintos datos también.
7. El Tiempo de Almacenamiento del fichero log debe estar incluido dentro de la ventana de disponibilidad de ambas fuentes.
8. Con el método log podemos tener todos los cambios de interés producidos, mientras que de la fuente con el método Timestamp sólo obtendremos algunos de los cambios producidos (dependiendo de cómo se realice la monitorización de la fuente). Por tanto podremos "integrar temporalmente" solo algunos de los cambios producidos en ambas fuentes.

6.3.3 Método TΔ con el resto de métodos

6.3.3.1 Trigger Delta – Trigger Delta

Los datos se extraen con el mismo método, Trigger Delta y se pueden obtener todos los cambios producidos en ambas fuentes. Es decir, podemos "integrar temporalmente" todos los cambios de interés producidos.

En este caso el resultado de los cruces es:

1. Será la aplicación operacional que necesite el mayor tiempo para escribir el cambio la que condicione la integración.
2. La fuente de datos que necesite más tiempo para realizar los cambios será la que determine el tiempo mínimo de espera para realizar la consulta de estas.
3. El tiempo de extracción a tener en cuenta a la hora de integrar los datos será el que implique un mayor tiempo de procesamiento en ambas fuentes.
4. Se podrá realizar un refresco de los datos del almacén cuando ambos ficheros delta sean accesibles.
5. El periodo de muestreo necesario para integrar las dos fuentes de datos deberá ser tal que coincida dentro de las ventanas de disponibilidad de ambas.
6. Si es el mismo dato: El TT de un cambio concreto recogido por ambas fuentes será el menor de las dos. Si son distintos datos también.
7. El tiempo de almacenamiento de las fuentes debe estar al menos incluido en la ventana de disponibilidad de ambas.
8. Se pueden obtener todos los cambios, puesto que ambas fuentes de datos pueden proveer de todos los cambios en los registros que nos interesan.

6.3.3.2 Trigger Delta – Trigger Directo

Disponemos de todos los cambios en ambas fuentes siendo posible "integrar temporalmente" todos ellos.

En este caso el resultado de los cruces es:

1. Será la aplicación operacional que necesite el mayor tiempo para escribir el cambio la que condicione la integración.
2. Sólo influirá el valor de la fuente de datos del tipo Triggered Delta en la integración, pues en el caso de la otra fuente de datos no es necesario consultarla.
3. Sólo influirá el valor de la fuente de datos del tipo Triggered Delta en la integración, pues en el caso de la otra fuente de datos no es necesario consultarla.
4. Se podrá realizar un refresco de los datos del almacén cuando el fichero delta sea accesible, pues no es necesario consultar la otra fuente de datos.
5. El proceso de extracción de los datos de la fuente del tipo triggered direct deberá producirse dentro de la ventana de disponibilidad de la fuente del tipo Timestamp para poder realizar la integración entre ambas.
6. Si es el mismo dato: El TT de un cambio concreto recogido por ambas fuentes será el menor de las dos. Si son distintos datos también.
7. El Tiempo de Almacenamiento del fichero delta debe estar incluido dentro de la ventana de disponibilidad de ambas fuentes.
8. Se pueden obtener todos los cambios, puesto que ambas fuentes pueden proveer de todos los cambios en los registros que nos interesan.

6.3.3.3 Trigger Delta – File Comparison

Podremos "integrar temporalmente" solo algunos de los cambios producidos dado que con File Comparison disponemos solo de algunos de los cambios. Respecto a la integración del parámetro "TT" no sería posible dado que el método File Comparison no dispone de dicho parámetro.

En este caso el resultado de los cruces es:

1. Será la aplicación operacional que necesite el mayor tiempo para escribir el cambio la que condicione la integración.
2. La fuente de datos que necesite más tiempo para modificar el fichero será la que determine el tiempo mínimo de espera para realizar la consulta de estas.
3. Estará determinado por el tiempo necesario en realizar la comparación de las imágenes de la fuente basada en este método de extracción, ya que generalmente es un proceso más lento que la consulta del fichero delta.
4. Se podrá realizar un refresco de los datos del almacén cuando ambas fuentes sean accesibles.
5. El periodo de muestreo necesario para integrar las dos fuentes de datos deberá ser tal que coincida dentro de las ventanas de disponibilidad de ambas.
6. No es posible integrar las fuentes de datos usando este valor porque la fuente del tipo "file comparison" no provee de esta información.
7. El tiempo de almacenamiento del fichero delta debe estar al menos incluido en la ventana de disponibilidad de ambas fuentes.
8. Con el método Trigger Directo podemos tener todos los cambios de interés producidos, mientras que de la fuente con el método File Comparison sólo obtendremos algunos de los cambios producidos (dependiendo de cómo se realice la monitorización de la fuente). Por tanto podremos "integrar temporalmente" solo algunos de los cambios producidos en ambas fuentes.

6.3.3.4 Trigger Delta – LOG

De ambas fuentes tenemos todos los cambios producidos. Por tanto podremos "integrar temporalmente" todos los cambios producidos en las fuentes.

En este caso el resultado de los cruces es:

1. Será la aplicación operacional que necesite el mayor tiempo para escribir el cambio la que condicione la integración.
2. La fuente de datos que necesite más tiempo para modificar el fichero será la que determine el tiempo mínimo de espera para realizar la consulta de estas.
3. El tiempo de extracción a tener en cuenta a la hora de integrar los datos será el que implique un mayor tiempo de procesamiento en ambas fuentes.

4. Se podrá realizar un refresco de los datos del almacén cuando ambas fuentes sean accesibles.
5. El periodo de muestreo necesario para integrar las dos fuentes de datos deberá ser tal que coincida dentro de las ventanas de disponibilidad de ambas.
6. Si es el mismo dato: El TT de un cambio concreto recogido por ambas fuentes será el menor de las dos. Si son distintos datos también.
7. El tiempo de almacenamiento de las fuentes debe estar al menos incluido en la ventana de disponibilidad de ambas.
8. Se pueden obtener todos los cambios, puesto que ambas fuentes pueden proveer de todos los cambios en los registros que nos interesan.

6.3.4 Método Trigger Directo con el resto de métodos

6.3.4.1 Trigger Directo – Trigger Directo

Se pueden tener todos los cambios de ambas fuentes, y a nivel abstracto podemos decir que se puede realizar la "integración temporal" de todos los parámetros temporales.

En este caso el resultado de los cruces es:

1. Será la aplicación operacional que necesite el mayor tiempo para escribir el cambio la que condicione la integración.
2. No influye en la integración, pues en ambos casos el valor de este parámetro es nulo.
3. No influye en la integración, pues en ambos casos el valor de este parámetro es nulo.
4. No influye en la integración, pues en ningún caso es necesario consultar las fuentes.
5. No es posible la integración entre estos dos tipos de fuentes porque el refresco se produce de manera asíncrona e individualmente, cuando se produce un cambio en alguna de las fuentes.
6. Si es el mismo dato: El TT de un cambio concreto recogido por ambas fuentes será el menor de las dos. Si son distintos datos también.
7. No es necesario tenerlo en cuenta, pues en ambas fuentes de datos se almacenan los datos indefinidamente.
8. Se pueden obtener todos los cambios, puesto que ambas fuentes pueden proveer de todos los cambios en los registros que nos interesan.

6.3.4.2 Trigger Directo – File Comparison

Con el método File Comparison solo obtendremos algunos cambios. Por tanto podremos "integrar temporalmente" solo algunos de los cambios producidos en ambas fuentes. La integración del parámetro "TT" no sería posible dado que el método File Comparison no dispone de dicho parámetro.

En este caso el resultado de los cruces es:

1. Será la aplicación operacional que necesite el mayor tiempo para escribir el cambio la que condicione la integración.
2. Sólo influirá el valor de la fuente de datos del tipo "file comparison" en la integración, pues en el caso de la otra fuente de datos no es necesario consultarla.
3. Estará determinado por el tiempo necesario en realizar la comparación de las imágenes de la fuente basada en este método de extracción, ya que en el caso de la otra fuente no es necesario consultarla para extraer los cambios.
4. Se podrá realizar un refresco de los datos del almacén cuando la fuente de datos del tipo File Comparison sea accesible, puesto que la otra no hace falta que sea consultada.
5. El proceso de extracción de los datos de la fuente del tipo triggered direct deberá producirse dentro de la ventana de disponibilidad de la fuente del tipo File Comparison para poder realizar la integración entre ambas.
6. No es posible integrar las fuentes de datos usando este valor porque la fuente del tipo File Comparison no provee de esta información.
7. No es necesario tenerlo en cuenta, pues en ambas fuentes de datos se almacenan los datos indefinidamente.
8. Con el método Trigger Directo podemos tener todos los cambios de interés producidos, mientras que de la fuente con el método File Comparison sólo obtendremos algunos de los cambios producidos (dependiendo de cómo se realice la monitorización de la fuente). Por tanto podremos "integrar temporalmente" solo algunos de los cambios producidos en ambas fuentes.

6.3.4.3 Trigger Directo – LOG

Ambos métodos proporcionan todos los cambios de interés producidos. Por tanto se puede llevar a cabo la "integración temporal" de todos los cambios producidos en las fuentes.

En este caso el resultado de los cruces es:

1. Será la aplicación operacional que necesite el mayor tiempo para escribir el cambio la que condicione la integración.
2. Sólo influirá el valor de la fuente de datos del tipo "file comparison" en la integración, pues en el caso de la otra fuente de datos no es necesario consultarla.
3. Estará determinado por el tiempo necesario en recorrer el fichero log y determinar los cambios producidos, ya que en el caso de la otra fuente no es necesario consultarla para extraer los cambios.
4. Se podrá realizar un refresco de los datos del almacén cuando el fichero log sea accesible, puesto que la otra fuente no hace falta que sea consultada.

5. El proceso de extracción de los datos de la fuente del tipo triggered direct deberá producirse dentro de la ventana de disponibilidad de la fuente del tipo File Comparison para poder realizar la integración entre ambas.
6. Si es el mismo dato: El TT de un cambio concreto recogido por ambas fuentes será el menor de las dos. Si son distintos datos también.
7. El Tiempo de Almacenamiento del fichero log debe estar incluido dentro de la ventana de disponibilidad de ambas fuentes.
8. Se pueden obtener todos los cambios, puesto que ambas fuentes pueden proveer de todos los cambios en los registros que nos interesan.

6.3.5 Método File Comparison con el resto de métodos

6.3.5.1 File Comparison – File Comparison

Con el método File Comparison podemos obtener solo algunos de los cambios producidos siendo solo posible realizar la "integración temporal" de algunos de los cambios producidos. Respecto a la integración del parámetro TT no sería posible dado que el método File Comparison no dispone de dicho parámetro.

En este caso el resultado de los cruces es:

1. Será la aplicación operacional que necesite el mayor tiempo para escribir el cambio la que condicione la integración.
2. La fuente de datos que necesite más tiempo para modificar el fichero será la que determine el tiempo mínimo de espera para realizar la consulta de estas.
3. El tiempo de extracción a tener en cuenta a la hora de integrar los datos será el que implique un mayor tiempo de procesamiento en ambas fuentes.
4. Se podrá realizar un refresco de los datos del almacén cuando ambos ficheros sean accesibles.
5. El periodo de muestreo debe coincidir dentro de la ventana de disponibilidad de ambos ficheros. Un valor alto puede implicar pérdida de movimientos en la fuente, mientras que uno pequeño puede producir una carga elevada de procesamiento sin ser realmente efectivo a la hora de detectar los cambios.
6. No es posible integrar las fuentes de datos usando este valor porque este tipo de fuente no provee de esta información.
7. No es necesario tenerlo en cuenta, pues en ambas fuentes de datos se almacenan los datos indefinidamente.
8. No es posible asegurar que se obtienen todos los cambios.

6.3.5.2 File Comparison – LOG

Con el método LOG todos los cambios y con File Comparison solo algunos, de lo que se deduce que podemos "integrar temporalmente" solo algunos de los cambios. La integración basándose en el parámetro TT no sería posible dado que el método File Comparison no dispone de dicho parámetro.

En este caso el resultado de los cruces es:

1. Será la aplicación operacional que necesite el mayor tiempo para escribir el cambio la que condicione la integración.
2. Será la aplicación operacional que necesite el mayor tiempo para escribir el cambio la que condicione la integración.
3. Estará determinado por el tiempo necesario en realizar la comparación de las imágenes que, generalmente, implicará un mayor tiempo de procesamiento que recorrer el fichero log y determinar los cambios producidos.
4. Se podrá realizar un refresco de los datos del almacén cuando ambas fuentes de datos sean accesibles.
5. El periodo de muestreo debe coincidir dentro de la ventana de disponibilidad de ambas fuentes.
6. No es posible integrar las fuentes de datos usando este valor porque la fuente del tipo File Comparison no provee de esta información.
7. El Tiempo de Almacenamiento del fichero log debe estar incluido dentro de la ventana de disponibilidad de ambas fuentes.
8. Con el método log podemos tener todos los cambios de interés producidos, mientras que de la fuente con el método File Comparison sólo obtendremos algunos de los cambios producidos (dependiendo de cómo se realice la monitorización de la fuente). Por tanto podremos "integrar temporalmente" solo algunos de los cambios producidos en ambas fuentes.

6.3.6 Método LOG con el resto de métodos

6.3.6.1 LOG – LOG

Se puede realizar la "integración temporal" de todos los cambios producidos en ambas fuentes ya que el método Log nos los proporciona.

En este caso el resultado de los cruces es:

1. Será la aplicación operacional que necesite el mayor tiempo para escribir el cambio la que condicione la integración.
2. Será la aplicación operacional que necesite el mayor tiempo para escribir el cambio la que condicione la integración.

3. El tiempo de extracción a tener en cuenta a la hora de integrar los datos será el que implique un mayor tiempo de procesamiento en ambas fuentes.
4. Se podrá realizar un refresco de los datos del almacén cuando ambos ficheros log sean accesibles.
5. El periodo de muestreo debe coincidir dentro de la ventana de disponibilidad de ambas fuentes.
6. Si es el mismo dato: El TT de un cambio concreto recogido por ambas fuentes será el menor de las dos. Si son distintos también.
7. El tiempo de almacenamiento de las fuentes debe estar al menos incluido en la ventana de disponibilidad de ambas.
8. Se pueden obtener todos los cambios, puesto que ambas fuentes pueden proveer de todos los cambios en los registros que nos interesan.

6.4 Algoritmos para la integración de las propiedades temporales

Uno de los problemas más importantes a la hora de construir el sistema es la resolución de las heterogeneidades existentes entre los diferentes sistemas operacionales componentes. En apartados anteriores se ha resuelto el problema de la heterogeneidad sintáctica, adoptando un modelo de datos canónico. Una vez resuelto el problema de la heterogeneidad sintáctica queda por resolver la heterogeneidad semántica.

En la figura 45 se muestra la arquitectura conjunta del procesador de integración y procesador del almacén de datos (Fase de integración en la figura figura 45), responsables de realizar el proceso de integración y obtener el esquema del almacén de datos.

A la arquitectura de referencia se le han añadido dos módulos necesarios para llevar a cabo la integración de las propiedades temporales de los datos teniendo en cuenta el método de extracción de datos utilizado, son el *Procesador de Integración Temporal* y el *Generador de Metadatos de Refresco*. Posteriormente el *Procesador de Refresco* se encargará de realizar la actualización del almacén.

Además, el procesador de integración y de almacenes de datos incluye los módulos necesarios para llevar a cabo la integración de los esquemas de datos. La figura permite observar como los módulos implicados en la integración temporal utilizan los resultados de los módulos involucrados en la integración de los esquemas de datos.

La metodología de integración consta de dos fases, basadas a su vez en las dos fases de las que consta la metodología de integración de esquemas de datos. Estas fases son: *Procesador de Integración Temporal* y *Generador de Metadatos de Refresco.*

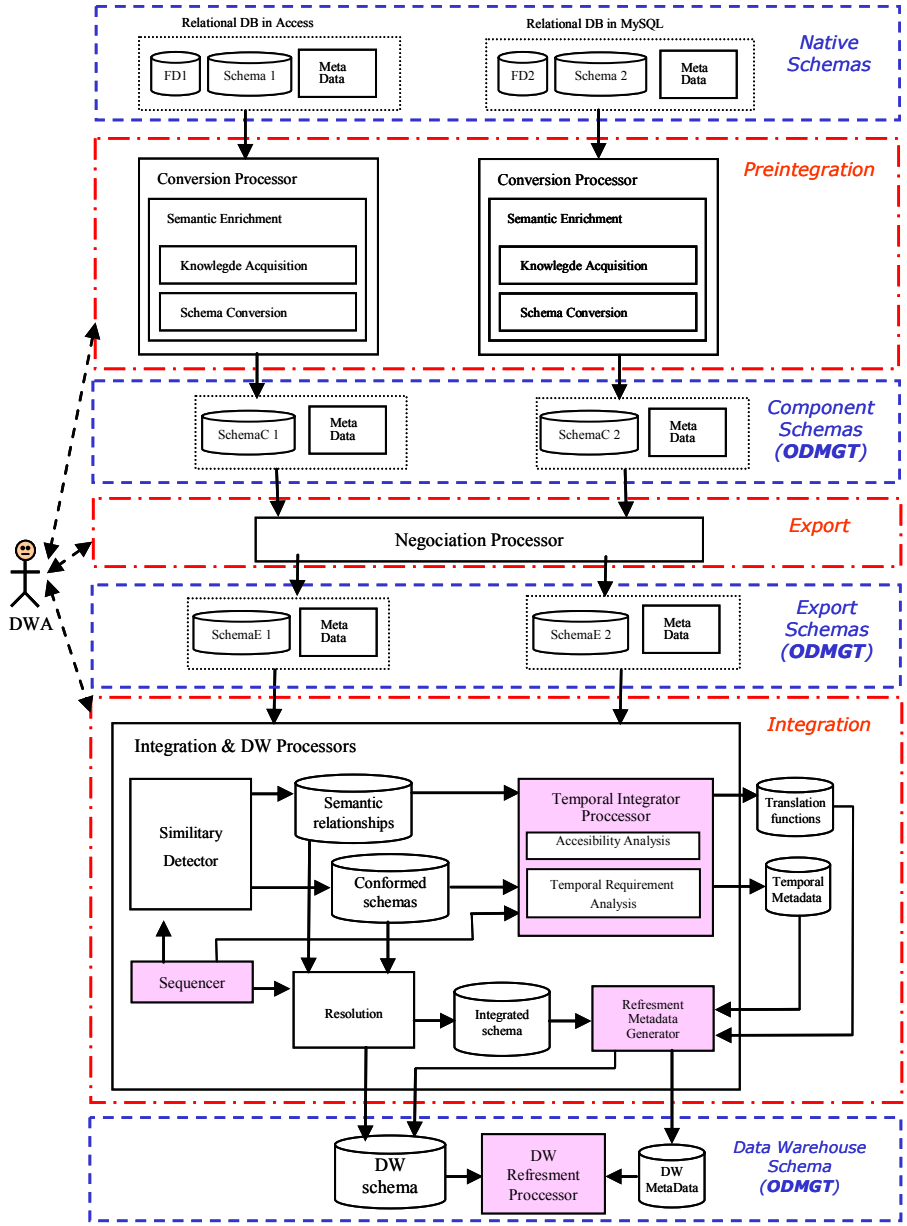

Figura 45. Arquitectura funcional

6.4.1 Fase I: Procesador de Integración Temporal

Esta fase utiliza el conjunto de relaciones semánticas y los esquemas conformados obtenidos durante la fase de detección de similitudes de la metodología de integración de esquemas de datos.

Se puede dividir a su vez en dos tareas diferentes: el *análisis de la accesibilidad* de ambas fuentes y el *análisis de requisitos temporales*. La primera de las tareas anteriores comprueba

que se satisfagan ciertos parámetros temporales comunes a cualquier tipo de fuente para que la integración pueda llevarse a cabo, mientras que la segunda de ellas, que sólo se llevaría a cabo en el caso de superar la primera fase, se centra en determinar si es posible la integración de dos fuentes de datos concretas. De esta forma se deducen las posibles relaciones semánticas entre los esquemas de los diferentes Sistemas Operacionales Componentes, facilitando su integración temporal. Estas dos tareas se describen con más detalle en secciones posteriores.

Como resultado se obtiene información en forma de reglas acerca de las posibilidades de integración que existen entre los datos provenientes de las fuentes (granularidad mínima, el periodo de refresco debe estar acotado entre algunos valores concretos,…). Esta información se guarda en el almacén de *Metadatos Temporales*. Como resultado del proceso de Integración Temporal se obtienen además una serie de *funciones de traslación*, que identifican los atributos de los esquemas de las fuentes de datos que se integran entre sí para obtener un atributo del esquema del almacén de datos.

Este proceso no asegura que pueda llevarse a cabo satisfactoriamente la integración de todos y cada uno de los cambios detectados en las dos fuentes de datos. Dependiendo de las características concretas de las fuentes involucradas en la integración puede que haya cambios que no puedan llegar a integrarse. Lo que si asegura este procedimiento es que el proceso de refresco de las fuentes de datos se va a realizar única y exclusivamente las veces que sean necesarias para conseguir los objetivos temporales propuestos por el diseñador del almacén.

En la figura 46 se muestra la información que se obtendría como resultado de aplicar el proceso de integración a dos atributos ("Name" y "User") de diferentes fuentes de datos (PGFederation y WebApplication) que se han encontrado semánticamente equivalentes, y dónde quedaría almacenada esta información.

Como *función de translación* se obtiene una regla que permite integrar dos valores de nombre diferentes encontrados en dos fuentes diferentes para un mismo piloto. En el almacén, el valor del parámetro *nombre* se integra seleccionando el valor de mayor longitud en las fuentes de datos, intentando de esta forma evitar introducir abreviaturas en la medida en que sea posible. Otro tipo de información que se obtendría son las *reglas temporales* que condicionan la integración.

En este ejemplo, debido al nivel de detalle con el que se almacenan los datos en ambas fuentes de datos, la *granularidad(GR)* que puede obtenerse para el dato, una vez integrado en el almacén de datos, tiene que ser, obligatoriamente, mayor que un día. Puesto que ambas fuentes pueden consultarse simultáneamente únicamente los lunes, el periodo de refresco podrá ser únicamente múltiplo de siete días (una vez a la semana, una vez cada dos semanas, una vez cada tres semanas, una vez al mes,…) y deberá encontrarse entre las cero y las veintitrés horas y cincuenta y nueve minutos del lunes.

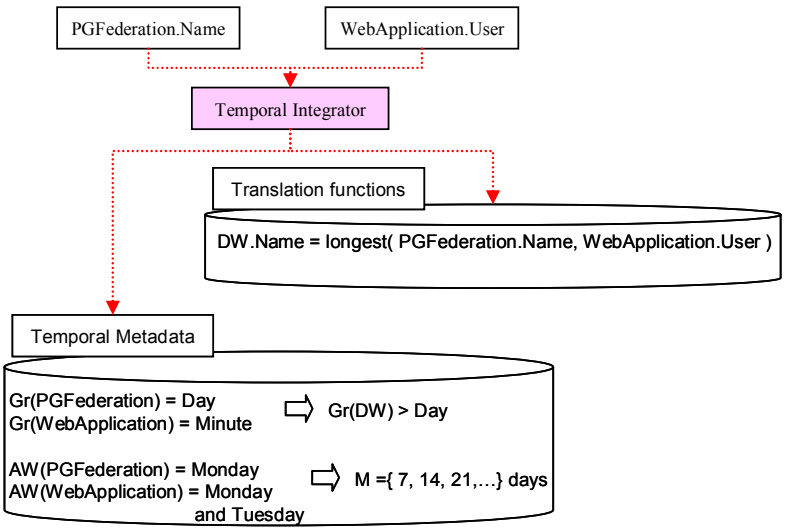

Figura 46. Integrador temporal

6.4.1.1 Análisis de accesibilidad

Se presenta a continuación un algoritmo determina un conjunto de instantes en los que ambas fuentes de datos son accesibles, teniendo en cuenta el nivel de detalle con el que se puede realizar la integración de los datos de ambas, que también se obtiene mediante la ejecución de este mismo algoritmo. Una vez determinados todos los posibles instantes en los que se puede realizar el refresco de las fuentes de datos, el siguiente proceso, denominado *Analizador de Requisitos Temporales*, descarta todos aquellos en los que no es posible realizar la integración debido a características particulares de las fuentes de datos concretas, que imponen una serie de restricciones temporales.

Dadas dos fuentes de datos, lo primero que se debe hacer es determinar la secuencia más pequeña en la que el conjunto de los valores de disponibilidad de ambas fuentes de datos se repite. Este concepto se denomina *patrón común*. Por ejemplo, si la ventana de disponibilidad de una fuente de datos re repite cada treinta y seis horas y la de otra cada veinticuatro, el patrón común será un intervalo de duración igual a setenta y dos horas. Como se puede ver en la figura 47, este intervalo puede comenzar en cualquier punto de la semana. Si la forma en la que se permite el acceso a las fuentes no sigue algún tipo de patrón (todos los días a una hora determinada, el tercer día del mes desde las siete horas a las quince horas y el quinto día del mes todo el día,…) no se puede determinar previamente si van a ser accesibles, por lo que debe comprobarse cada vez que se vaya a realizar el refresco de los datos.

El algoritmo primero determina el máximo nivel de detalle con el que ambas fuentes de datos pueden proporcionar datos. Por ejemplo, si una fuente proporciona datos con un nivel de detalle de un día, mientras que otra los proporciona a nivel de hora, no será

posible integrarlas de forma que obtengamos un nivel de detalle superior a un día (horas, minutos, segundos,…). La función MínimoDetalle determina este valor.

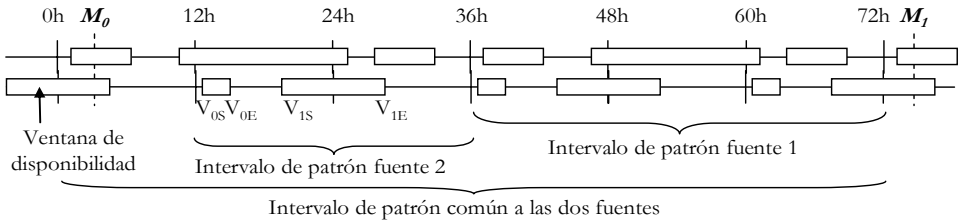

Figura 47. Dos fuentes y la ventana de disponibilidad

Puede ocurrir que la unidad (el grano) del nivel de detalle que se puede obtener tras la integración de ambas fuentes de datos tenga una longitud mayor que el patrón común de disponibilidad. Por ejemplo, que se pueda obtener una granularidad a nivel de día y la longitud del patrón común sea de varias horas. En este caso, bastaría con consultar las fuentes de datos una sola vez al cabo de un día (por cuestión de eficiencia, no tiene sentido refrescar una fuente de datos más a menudo de lo que se va a almacenar). Por lo tanto, se establece que el valor del periodo de refresco podrá ser como máximo la longitud de la unidad del nivel de detalle, obtenida mediante la función intervalo en el algoritmo. El valor del periodo de muestreo podrá ser, en el caso del ejemplo anterior, múltiplo de un día (dos días, tres días, una semana,…). Dentro del patrón común se escoge para realizar el refresco el instante en el comienza el intervalo de mayor longitud en el que ambas fuentes se encuentran disponibles, de forma que haya mayor probabilidad para satisfacer las restricciones impuestas en la segunda fase del módulo de integración temporal (*Análisis de Requisitos Temporales*). Este intervalo lo determina la función IntervaloVDdeMayorLongitud. Sólo hace falta establecer el primer instante en el que se debe realizar el refresco (M_0 en el algoritmo), pues el resto se puede obtener a partir de éste, sumándole tantas veces la longitud del intervalo de refresco como sean necesarias (por ejemplo, $M_4 = M_0 + 4*M$).

En el caso de que la unidad (el grano) del nivel de detalle que se puede obtener tras la integración de ambas fuentes de datos tenga una longitud menor que el patrón común de disponibilidad es necesario determinar en que instantes dentro del patrón común ambas fuentes de datos van a estar disponibles para poder realizar un refresco de sus datos. Puesto que no tiene sentido refrescar un dato más a menudo de lo que va a ser almacenado, sólo se seleccionan aquellos entre los que transcurre la longitud de la unidad de granularidad con la que se va a obtener el dato una vez integrado. Por ejemplo, si la granularidad con la que va a integrar el dato es de segundo, los instantes estarán distanciados temporalmente un segundo (eliminando aquellos en los que no se puede acceder a alguna de las fuentes). Se comprueba entonces que, para todos los instantes del patrón común separados a una distancia igual al intervalo de la unidad de granularidad, ambas fuentes de datos son accesibles. Si es así se añade el instante correspondiente al

vector de instantes en los que se puede realizar el refresco (M_i). Algunos de los instantes incluidos en este vector serán descartados en la siguiente fase por no cumplir alguno de los requisitos específicos que dependen de las fuentes concretas.

El algoritmo es el siguiente:

```
Si la disponibilidad de las fuentes de datos es periódica
        GrMaxima = MinimoDetalle(Granularidad fuente1, Granularidad fuente2)
        // Ejemplo: día = MinimoDetalle(hora, día)
        Si intervalo(GrMaxima) >= intervalo del patrón común
                MayorVD = IntervaloVDdeMayorLongitud(PatronComun)
                M0 = MayorVDinicio
                M = intervalo(GrMaxima)
        Si-no
                i = 0
                j = 0
                Mientras intervalo(GrMaxima)*j < Intervalo(PatronComun).fin
                        Si ambas fuentes son accesibles en intervalo(GrMaxima)*j
                                Mi = intervalo(GrMaxima)*i
                                i++
                        j++
Si-no
        No es posible determinar si se pueden integrar
```

6.4.1.2 Análisis de requisitos temporales

Una vez obtenido un conjunto de instantes en los que las fuentes pueden ser consultadas de forma conveniente de acuerdo a la granularidad y accesibilidad de las mismas, el *analizador de requisitos* comprueba que cada uno de estos instantes obtenidos en la fase anterior (en los que se pretende realizar el refresco de los datos), satisfacen una serie de requisitos mínimos. En este proceso si que es determinante el tipo de fuentes que van a ser integradas, porque cada una de ellas impondrá unos requisitos diferentes. En la Tabla 6 se resumen los requisitos que debe cumplir cada tipo de fuente en función del tipo de fuente con el que va a ser integrado.

	Application assisted (AA)	Timestamp based (TS)	Triggered "delta" (TΔ)	Triggered "direct" (TD)	File Comparison (FC)	Log (LOG)
Application Assisted (AA)	TA(AA1)>M TA(AA1)>M	TA(AA)>M	TA(AA)>M TA(TΔ)>M	-	TE(FC)<M TA(AA)>M Mi+TE(FC)<VDfin	TA(AA)>M TA(LOG)>M
Timestamp Based (TS)		-	TA(TΔ)>M	-	TE(FC)<M Mi+TE(FC)<VDfin	TA(LOG)>M
Triggered "Delta" (TΔ)			TA(TΔ1)>M TA(TΔ2)>M	-	TE(FC)<M Mi+TE(FC)<VDfin	TA(TΔ)>M TA(LOG)>M
Triggered "Direct" (TD)				-	-	-
File Comparison (FC)					TE(FC1)+ TE(FC2)<M Mi+TE(FC1)+TE(FC2)<VDfin	TE(FC)<M TA(LOG)>M Mi+TE(FC)<VDfin
Log (LOG)						TA(LOG1)>M TA(LOG2)>M

Tabla 6. Requisitos de las fuentes

A continuación se detallan los pasos a seguir para cada uno de los cruces recogidos en la tabla anterior.

6.4.1.2.1 Application Assisted con el resto de métodos

Application Assisted – Application Assisted

En este caso se debe comprobar que el tiempo de almacenamiento de los cambios en ambas fuentes sea superior, al menos, al periodo de refresco, de forma que no se pierdan cambios entre refrescos consecutivos.

Para cada Mi
    Si TA(AA1) < Mi+1 - Mi | TA(AA2) < Mi+1 - Mi
        Mi = NULL

Application Assisted – Timestamp

Puesto que la fuente de datos de tipo Timestamp no almacena temporalmente los cambios, en este caso sólo hay que comprobar que la fuente del tipo Application Assisted si los guarda, al menos, entre un refresco y el siguiente.

Para cada Mi
    Si TA(AA) < Mi+1 - Mi
        Mi = NULL

Application Assisted - Trigger Delta

Se trata de la misma situación que se daba en el caso de la integración de dos fuentes de tipo Application Assisted, pues en ambos casos se debe realizar la integración entre dos fuentes que generan ficheros delta.

```
Para cada Mi
        Si TA(AA) < Mi+1 - Mi | TA(TΔ) < Mi+1 - Mi
                Mi = NULL
```

Application Assisted – Trigger Directo

La fuente de tipo Trigger Directo realiza el proceso de integración de manera asíncrona (cuando se lanza el trigger), por lo que no es posible determinar previamente si se va a poder llevar a cabo en función de los parámetros que se tratan en este apartado.

Application Assisted – File Comparison

Se debe comprobar primero, como en todos los casos donde los cambios se almacenen temporalmente que éstos se almacenan el tiempo suficiente para que sean tenidos en cuenta antes de ser eliminados. A continuación se comprueba que el tiempo necesario para extraer el cambio de la fuente de tipo File Comparision (que normalmente requiere un mayor tiempo de cómputo) sea menor que el tiempo que transcurre entre dos refrescos consecutivos. De otra manera no se podría realizar el segundo refresco por estar aún intentando detectar si se ha producido un cambio en el refresco anterior. Por último, se debe comprobar, para cada uno de los refrescos que se producen dentro del patrón común, que la fuente de datos del tipo File Comparision es accesible durante todo el proceso de extracción del cambio

```
Para cada Mi
        Si TA(AA) < Mi+1 - Mi
                Mi = NULL
        Si TE(FC) > Mi+1 – Mi
                Mi = NULL
        ventanaDeDisponibilidad = IntervaloVentanaDisponibilidadEn(FC, TT(Mi))
        Si Mi + TE(FC) > ventanaDeDisponibilidad.Fin | ventanaDeDisponibilidad = NULL
                Mi = NULL
```

Application Assisted – LOG

Se trata de la misma situación que se daba en el caso de la integración de dos fuentes de tipo Application Assisted.

```
Para cada Mi
        Si TA(AA) < Mi+1 - Mi | TA(LOG) < Mi+1 - Mi
                Mi = NULL
```

6.4.1.2.2 Timestamp con el resto de métodos

Timestamp – Timestamp

No existen reglas específicas para integrar este tipo de fuente entre sí.

Timestamp - Trigger Delta

Puesto que la fuente de datos de tipo Timestamp no almacena temporalmente los cambios, en este caso sólo hay que comprobar que la fuente del tipo Trigger Delta si los guarda, al menos, entre un refresco y el siguiente.

```
Para cada Mi
        Si TA(TΔ) < Mi+1 - Mi
                Mi = NULL
```

Timestamp – Trigger Directo

La fuente de tipo Timestamp realiza el proceso de integración de manera asíncrona (cuando se lanza el trigger), por lo que no es posible determinar previamente si se va a poder llevar a cabo en función de los parámetros que se tratan en este apartado.

Timestamp – File Comparison

Se debe comprobar primero que el tiempo necesario para extraer el cambio de la fuente de tipo File Comparision (que normalmente requiere un mayor tiempo de cómputo) sea menor que el tiempo que transcurre entre dos refrescos consecutivos. De otra manera no se podría realizar el segundo refresco por estar aún intentando detectar si se ha producido un cambio en el refresco anterior. A continuación, se debe comprobar, para cada uno de los refrescos que se producen dentro del patrón común, que la fuente de datos del tipo File Comparision es accesible durante todo el proceso de extracción del cambio

```
Para cada Mi
        Si TE(FC) > Mi+1 – Mi
                Mi = NULL
        ventanaDeDisponibilidad = IntervaloVentanaDisponibilidadEn(FC, TT(Mi))
        Si Mi + TE(FC) > ventanaDeDisponibilidad.Fin | ventanaDeDisponibilidad = NULL
                Mi = NULL
```

Timestamp – LOG

Se debe comprobar que el tiempo de almacenamiento de los cambios registrados en el fichero log es superior al valor del periodo de refresco para que sean tenidos en cuenta, al menos una vez.

```
Para cada Mi
```

```
Si TA(LOG) < Mi+1 - Mi
        Mi = NULL
```

6.4.1.2.3 Trigger Delta con el resto de métodos

Trigger Delta – Trigger Delta

En este caso se debe comprobar que el tiempo de almacenamiento de los cambios en ambas fuentes sea superior, al menos, al periodo de refresco, de forma que no se pierdan cambios entre refrescos consecutivos.

```
Para cada Mi
        Si TA(TΔ1) < Mi+1 - Mi | TA(TΔ2) < Mi+1 - Mi
        Mi = NULL
```

Trigger Delta – Trigger Directo

La fuente de tipo Trigger Delta realiza el proceso de integración de manera asíncrona (cuando se lanza el trigger), por lo que no es posible determinar previamente si se va a poder llevar a cabo en función de los parámetros que se tratan en este apartado.

Trigger Delta – File Comparison

Se trata de la misma situación que se daba en el caso de la integración entre fuentes de tipo Application Assisted y File Comparision.

```
Para cada Mi
        Si TA(TΔ) < Mi+1 - Mi
                Mi = NULL
        Si TE(FC) > Mi+1 – Mi
                Mi = NULL
        ventanaDeDisponibilidad = IntervaloVentanaDisponibilidadEn(FC, TT(Mi))
        Si Mi + TE(FC) > ventanaDeDisponibilidad.Fin | ventanaDeDisponibilidad = NULL
                Mi = NULL
```

Trigger Delta – LOG

Se trata de la misma situación que se daba en el caso de la integración de dos fuentes de tipo Application Assisted.

```
Para cada Mi
        Si TA(LOG) < Mi+1 - Mi | TA(TΔ) < Mi+1 - Mi
                Mi = NULL
```

6.4.1.2.4 Trigger Directo con el resto de métodos

Trigger Directo – Trigger Directo

La fuente de tipo Trigger Directo realiza el proceso de integración de manera asíncrona (cuando se lanza el trigger), por lo que no es posible determinar previamente si se va a poder llevar a cabo en función de los parámetros que se tratan en este apartado.

Trigger Directo – File Comparison

La fuente de tipo Trigger Directo realiza el proceso de integración de manera asíncrona (cuando se lanza el trigger), por lo que no es posible determinar previamente si se va a poder llevar a cabo en función de los parámetros que se tratan en este apartado.

Trigger Directo – LOG

La fuente de tipo Trigger Directo realiza el proceso de integración de manera asíncrona (cuando se lanza el trigger), por lo que no es posible determinar previamente si se va a poder llevar a cabo en función de los parámetros que se tratan en este apartado.

6.4.1.2.5 File Comparison con el resto de métodos

File Comparison – File Comparison

Es este tipo de fuentes influye de manera determinante el tiempo necesario para extraer los cambios producidos. Se debe comprobar primero que el tiempo empleado en realizar esta operación en las dos fuentes sea inferior al valor del periodo de refresco. Si no es así, cuando se intente realizar el siguiente refresco aún se estará llevando a cabo la detección de cambios en alguna de las fuentes. Si se satisface esta condición hay que comprobar que ambas fuentes son accesibles durante la extracción de los cambios. Si la extracción de estos cambios se realiza de manera secuencial (primero de una fuente y luego de otra) se puede optimizar esta comprobación variando el orden en que se consultan las fuentes en función de la longitud del intervalo que están disponibles a partir del momento en que se refrescan (para cada uno de los refrescos del patrón común). Por simplicidad del algoritmo no se ha tenido en cuenta esta última situación.

```
Para cada Mi
        Si TE(FC1) < Mi+1 - Mi
                Mi = NULL
        Si TE(FC2) > Mi+1 – Mi
                Mi = NULL
        ventanaDeDisponibilidad1 = IntervaloVentanaDisponibilidadEn(FC1, TT(Mi))
        ventanaDeDisponibilidad2 = IntervaloVentanaDisponibilidadEn(FC2, TT(Mi))
        ventanaDeDisponibilidad = Interseccion(FC1, FC2)
```

```
        Si Mi + TE(FC1) > ventanaDeDisponibilidad.Fin | Mi + TE(FC2) >
        ventanaDeDisponibilidad.Fin | ventanaDeDisponibilidad = NULL

        Mi = NULL
```

File Comparison – LOG

Se trata de la misma situación que se da a la hora de integrar fuentes de tipo Application Assisted y File Comparision.

```
Para cada Mi
        Si TA(LOG) < Mi+1 - Mi
                Mi = NULL
        Si TE(FC) > Mi+1 – Mi
                Mi = NULL
        ventanaDeDisponibilidad = IntervaloVentanaDisponibilidadEn(FC, TT(Mi))
        Si Mi + TE(FC) > ventanaDeDisponibilidad.Fin | ventanaDeDisponibilidad = NULL
                Mi = NULL
```

6.4.1.2.6 LOG con el resto de métodos

LOG – LOG

En este caso se debe comprobar que el tiempo de almacenamiento de los cambios en ambas fuentes sea superior, al menos, al periodo de refresco, de forma que no se pierdan cambios entre refrescos consecutivos.

```
Para cada Mi
        Si TA(LOG) < Mi+1 - Mi | TA(LOG) < Mi+1 - Mi
                Mi = NULL
```

6.4.2 Fase II: Generador de Metadatos de Refresco

Durante la segunda fase se determinan los parámetros más adecuados para llevar a cabo el refresco de los datos del Esquema Integrado, generado a partir de la fase de resolución de la metodología de integración de esquemas de datos. Es en esta segunda fase donde, a partir de los requisitos mínimos generados por la primera fase de integración temporal y que se recogen en el almacén de metadatos temporales, el diseñador del almacén fija los parámetros de refresco de los datos almacenados en el almacén. Como resultado se obtiene el esquema del almacén de datos junto con los metadatos necesarios para ser refrescado de forma coherente en función del tipo de método de extracción y otras características temporales de las fuentes de datos a las que se accede. Los metadatos de refresco generados se sitúan en el almacén de metadatos del almacén para que éste realice el refresco de los datos de acuerdo con ellos.

Este procedimiento se puede llevar a cabo de forma manual o automática. La integración manual consiste en que alguna persona (normalmente el administrador del almacén) se

encarga de determinar los parámetros temporales del proceso de integración que se debe llevar a cabo. El sistema comprueba que tales parámetros son coherentes dadas las características de las fuentes de datos de las que se va a extraer la información e informa del resultado.

6.4.3 Integración semi-automática

La integración automática implica disponer de alguna herramienta informática que realiza la elección de los parámetros de integración de forma automática atendiendo a algún criterio (máximo detalle, menor tiempo de computo,…). Este tipo de integración tiene la ventaja de descargar a una persona del trabajo tedioso de encontrar los parámetros que se ajusten de manera adecuada a los objetivos que se pretenden conseguir. Además, dado el tipo de sistema que se desea construir, normalmente se buscará obtener el máximo nivel de detalle en los datos almacenados, aunque luego se presente a los usuarios adaptándolos al nivel de detalle que necesiten individualmente. A pesar de esta clara ventaja no es fácil obtener una herramienta de este tipo, por lo que se ha optado por construir una solución semi-automática que resuelve el problema de la detección de las relaciones semánticas y presenta al diseñador del almacén los criterios mínimos que debe seguir la integración de dos parámetros de diferentes fuentes. Una vez que el diseñador del almacén introduce el valor de estos parámetros la aplicación comprueba su validez e informa al mismo.

6.4.4 Procesador de Refresco

A continuación de la integración temporal y una vez obtenido el esquema del almacén, es necesario el mantenimiento y actualización del mismo. Esta función la realiza el *Procesador de Refresco del Almacén*.

A partir de los requisitos mínimos que se deben cumplir para llevar a cabo la integración entre dos datos de diferentes fuentes de datos, obtenidos mediante el módulo de Integración Temporal y el esquema integrado obtenido por el módulo de resolución se fijan los parámetros de refresco de los datos almacenados en el almacén. Este proceso puede llevarse a cabo con la ayuda de la herramienta descrita en la sección 6.4.3. Estos parámetros se almacenan en forma de metadatos en el esquema del almacén y serán utilizados por el *Procesador de Refresco* para actualizar el almacén con los datos producidos en las fuentes. El Capítulo 7 explica con más detalle las tareas realizadas por el procesador.

6.5 Ejemplo ilustrativo

Supongamos que necesitamos integrar el parámetro *nivel*, que se refiere a la habilidad de los pilotos para aprovechar las características de la atmósfera [Araq06]. En la fuente de datos de la federación autonómica de parapente sólo disponemos de esta información para los instructores. Estos se clasifican en seis niveles diferentes dependiendo de la

experiencia y otras características. Para poder enviar sus vuelos a través de Internet a la aplicación es necesario que se inscriban en la misma e incluyan sus datos personales. Puesto que esta segunda fuente de datos se deben tener en cuenta a todos los pilotos, los niveles de seguridad se dividen en tres categorías (los principiantes, los intermedios y los avanzados), teniendo en cada una de ellas diferentes subniveles. Conforme avanza el tiempo los pilotos van aumentando sus habilidades y presentándose a pruebas para subir de nivel. En el caso de los pilotos que también son instructores esta información se encuentra duplicada y debe ser integrada. La integración semántica obtiene como resultado una fórmula para convertir valores en las diferentes escalas de las fuentes de datos a la escala del parámetro integrado en el almacén, de la forma en que se muestra en la figura x, mientras que la integración temporal impone unas restricciones acerca de cuando puede producirse esta integración. En este caso concreto, las reglas que permiten la integración de dos valores diferentes, pero semánticamente equivalentes, encontrados en fuentes diferentes para un mismo piloto serían:

- Nombre: en el almacén de datos, el valor del parámetro *nombre* se integra seleccionando el valor de mayor longitud en las fuentes de datos, intentando de esta forma evitar introducir abreviaturas en la medida en que sea posible.
- Nivel: los valores susceptibles de ser integrados en este caso son aquellos correspondientes a los instructores de la federación, que se consideran pilotos avanzado en la escala de la aplicación Web. La función de traslación en este caso realiza un ajuste de los valores recogidos en las fuentes de datos a una nueva escala de niveles creada para este atributo en el almacén de datos.
- Fecha: en este caso, la regla de translación consiste en seleccionar aquella fecha que corresponde con el instante más antiguo (si el diseñador del almacén lo hubiese estimado más oportuno podría haber elegido hacer la media entre los dos valores o cualquier otro tipo de cálculo).

6.5.1 Ejemplo de LOG – LOG

Supongamos que ambas fuentes de datos utilizan un método de extracción basado en ficheros log para extraer los cambios producidos.

En la primera de las fuentes de datos se apunta en la ficha del piloto el día en que se produce un cambio de nivel (ej: 17-Agosto-97). El fichero log de la federación que recoge estos cambios tiene una marca temporal con un nivel de detalle diario, esto es, que sólo apunta el día en que se almacena en el log (no se guarda la hora). La granularidad del parámetro *nivel* de esta primera fuente (y de todos sus parámetros) es un día, porque es el máximo nivel de detalle con el que podemos obtener esta información.

En cambio, la segunda de las fuentes de datos no requiere disponer de un nivel de detalle tan exacto, por lo que en la información de registro del piloto sólo pide que se indique el mes en el que obtuvo el nuevo nivel (ej: Agosto-97). En cambio, la información en el log

de esta segunda fuente de datos tiene asociada una marca temporal con un nivel de detalle de un minuto, es decir, que apunta la hora en que se almacenan los cambios en el fichero log. La granularidad de este parámetro en la segunda fuente es un minuto, a pesar de que en la ficha del piloto sólo se apunte el mes en el que se produce el cambio, porque a efectos del método de extracción del cambio todos los valores recogidos en la ficha del piloto son campos de datos de usuario.

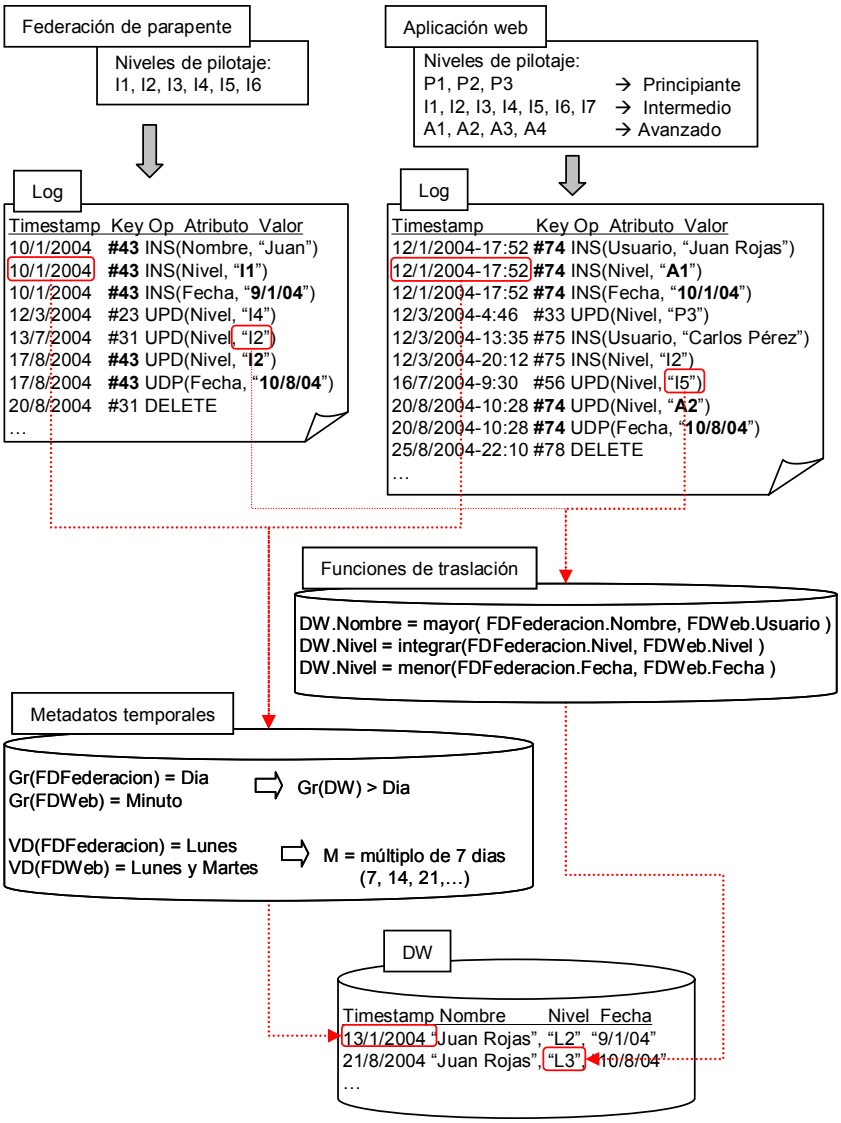

Figura 48. Dos fuentes con el método LOG

El máximo nivel de detalle con el que se puede obtener el parámetro *nivel* integrado (y el resto de los atributos) es un día, esto es, el máximo nivel de detalle disponible en ambas

fuentes de datos. En el almacén de metadatos temporales se introduce una regla que indica esta circunstancia. Esta regla impone, además, una restricción en el proceso de refresco, pues por cuestión de eficiencia no tiene sentido refrescar las fuentes de datos en busca de cambios producidos más a menudo del nivel de detalle con el que vayan a ser almacenadas en el almacén, por lo que, en este caso, no tiene sentido hacerlo más de una vez al día. Además, puesto que ambas fuentes pueden consultarse simultáneamente únicamente los lunes, el periodo de refresco podrá ser únicamente múltiplo de siete días (una vez a la semana, una vez cada dos semanas, una vez cada tres semanas, una vez al mes,…) y deberá encontrarse entre las cero y las veintitrés horas y cincuenta y nueve minutos del lunes. Podemos ver lo comentado antes en la figura 48.

Aunque sabemos que dado un piloto cualquiera su nivel de pilotaje no va a cambiar antes de seis meses (es el periodo mínimo de tiempo que debe esperar un piloto para presentarse a una prueba para aumentar de nivel), si que tiene sentido consultar la fuente diariamente, pues puede que otros pilotos si que hayan registrado un cambio.

6.5.2 Ejemplo de LOG – FC

La granularidad es un parámetro que viene impuesto por la fuente de datos, mientras que el periodo de refresco depende del diseñador del almacén. Esto es así en todos los casos salvo en las fuentes de tipo File Comparison, en las que el nivel de detalle de los cambios producidos en ellas viene impuesto por el intervalo que transcurre entre una comparación de imágenes y la siguiente.

Supongamos que la información de la segunda fuente de datos del ejemplo anterior, correspondiente al registro de usuarios en la aplicación Web, es accesible ahora por medio de una página Web. En este caso habría que aplicar un método de extracción basado en comparación de imágenes (File Comparison). En la figura 49 se muestra esta nueva situación. El programa encargado de extraer los cambios producidos entre dos estados consecutivos es el responsable ahora de señalar el instante en el que se detectan los cambios. El nivel de detalle (granularidad) con el que se marcan temporalmente los datos no podrá ser inferior a una hora, pues no sabemos en que minuto exacto se han producido. Tampoco tiene sentido utilizar un nivel de detalle mayor (día, por ejemplo), pues podría darse el caso de detectar dos cambios a los que les correspondiese el mismo valor de la marca temporal, lo que no sería correcto.

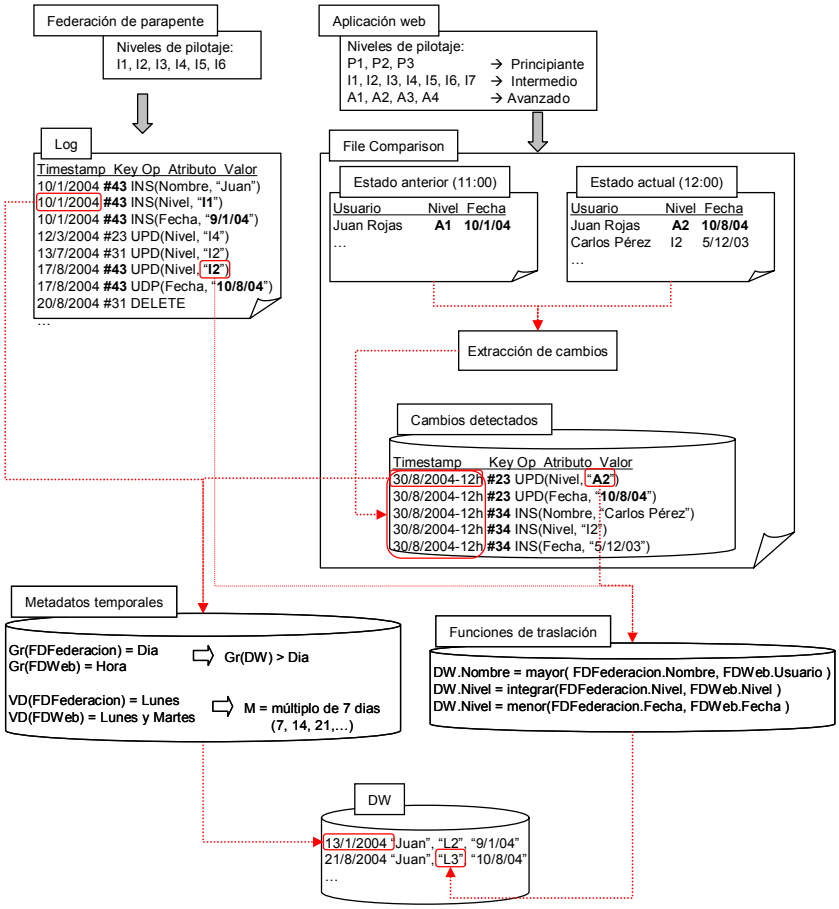

Figura 49. Dos fuentes con los métodos LOG y FC

6.5.3 Ejemplo de LOG – TS

Otra de las situaciones posibles sería aquella en la que la segunda de las fuentes de datos, correspondiente a la información de registro de los usuarios de la aplicación Web, estuviese organizada de forma que cada vez que se produjese un cambio se apuntara el instante en el que se produjo. Para extraer los cambios en este caso podría utilizarse el método TS (Time Stamp based).

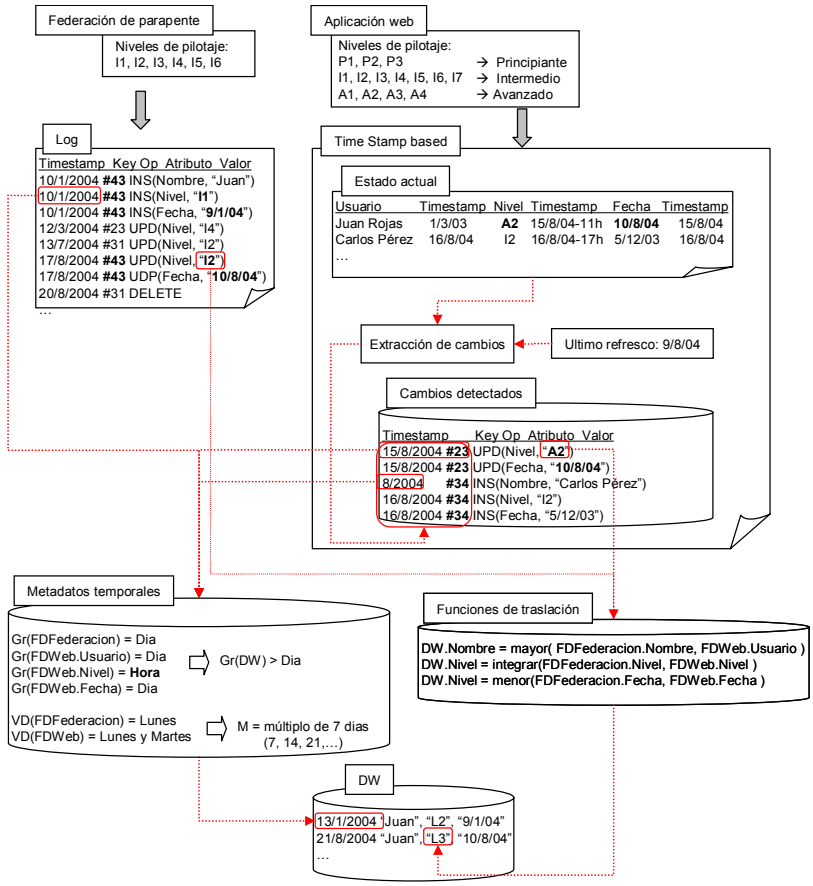

Figura 50. Dos fuentes con los métodos LOG y TS

Puede ocurrir que, como se muestra en la figura 50, en lugar de utilizar una sola marca temporal para el registro completo, se marque individualmente cada uno de los atributos. En el ejemplo, el atributo *nivel* esta marcado con un nivel de detalle de hora, mientras que el resto está marcado a nivel de día. Por tanto, en el almacén de metadatos temporales hay que distinguir entre cada uno de los parámetros. En este ejemplo no afecta a la integración el hecho de que el parámetro *nivel* tenga una granularidad de minuto, porque ésta queda condicionada por la granularidad de la fuente que provee este mismo atributo con menor detalle, que en este caso es de día.

6.6 Conclusiones

En este capítulo se ha propuesto la arquitectura de integración necesaria para realizar la integración de las propiedades temporales de los datos. Se han definido los módulos implicados en la integración y se han encuadrado dentro de nuestra arquitectura funcional previamente utilizada en bases de datos federadas. Estos módulos son los encargados de

comprobar que se satisfagan ciertos parámetros temporales para que la integración pueda llevarse a cabo y si es posible la integración de dos fuentes de datos concretas. Como resultado se obtiene información en forma de reglas acerca de las posibilidades de integración que existen entre los datos provenientes de las fuentes (granularidad mínima, el periodo de refresco debe estar acotado entre algunos valores concretos,…). Se han propuesto también los algoritmos adecuados para realizar la integración y se ha ilustrado el proceso con un ejemplo.

Capítulo 7

OPERACIÓN DEL SISTEMA

Una vez que se ha decidido qué datos van a materializarse en el alamacén de datos, es necesario llevar a cabo la actualización del mismo con el objeto de mantenerlo consistente con los datos almacenados en las fuentes. Esta actualización se realiza de acuerdo a las decisiones tomadas por el Diseñador del almacén de datos. El proceso para llevar a cabo el refresco incluye, definir qué tareas deben ejecutarse durante el proceso, extraer los datos de las fuentes, transformarlos a un formato común y cargarlos en el almacén. En este capítulo se explica como se llevaría a cabo el refresco del almacén de datos enmarcando este proceso dentro de nuestra arquitectura. Se definen los módulos necesarios y se presentan los algoritmos para realizar el refresco.

El capítulo se estructura como sigue. En la sección 7.1 se introduce el problema del refresco del almacén. En la sección 7.2 se presenta nuestra arquitectura de refresco y se explican los módulos de la que está compuesta. En la sección 7.3 se muestran los algoritmos de refresco propuestos. En la sección 7.4 se muestra un ejemplo ilustrativo de utilización de la arquitectura de operación para realizar el refresco. Y finalizamos con las conclusiones del capítulo en la sección 7.5.

7.1 Introducción

Como hemos visto en capítulos anteriores, el desarrollo de los almacenes de datos compromete a arquitecturas, algoritmos, modelos y herramientas con el objetivo de integrar datos provenientes de diferentes sistemas operacionales para proporcionar información a los sistemas de ayuda de decisión. Un sistema de almacenes de datos incluye el almacén de datos en sí mismo y todos los componentes responsables de la construcción, el acceso y el refresco del almacén de datos [Vavo99].

La figura 51 nos muestra un entorno típico de almacén de datos. Los datos provenientes de los sistemas operacionales se extraen de dichos sistemas para ser integrados en el almacén de datos (fase de adquisición). En el almacén de datos se guardan los datos

extraídos conforme al modelo de datos elegido, y los data marts almacenan datos provenientes del almacén de datos utilizando el modelo de datos multidimensional (fase de almacenamiento). Finalmente, los usuarios acceden al almacén de datos o a los data marts utilizando varias herramientas y aplicaciones para la toma de decisiones (fase de análisis).

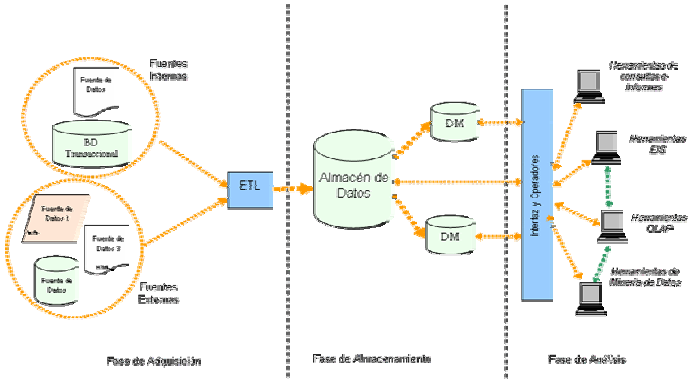

Figura 51. Arquitectura de un almacén de datos

Implantar un almacén de datos concreto es una tarea compleja que conlleva dos fases principales. En la fase de configuración el diseñador del almacén determina (siguiendo los requerimientos de información de los usuarios) los sistemas operacionales de los que se extraerán los datos, los datos a extraer de dichos sistemas, los métodos de extracción a utilizar, las transformaciones necesarias para la integración de los datos, y las herramientas que se utilizarán durante la fase de análisis. Después de la carga inicial (la primera carga de datos en el almacén de acuerdo a la configuración), durante la fase de operación el almacén debe ser refrescado regularmente para reflejar los cambios producidos en los sistemas operacionales desde el último refresco (los data marts también son refrescados a partir de los datos disponibles en el almacén de datos).

7.2 Refresco del almacén de datos

A continuación damos una visión de las tareas relacionadas con el proceso de refresco del almacén de datos. Las cuestiones a tratar incluyen: las tareas que se deben realizar durante el refresco, los procesos de refresco incremental del almacén, y las cuestiones temporales relacionadas con el inicio del proceso de refresco.

La primera cuestión se refiere a qué tareas se deben ejecutar durante el refresco. Los datos operacionales se encuentran en una amplia variedad de sistemas que se ejecutan en diversas plataformas y que utilizan diversos formatos. Debido a la heterogeneidad de la fuentes de datos, el proceso de refresco implica extraer los datos de diferentes fuentes y transformarlos a un formato común (el modelo de datos canónico del almacén). Los datos de los sistemas operacionales a menudo contienen errores, son inconsistentes o

incompletos. Por tanto, es necesario realizar una limpieza antes de que sean cargados en el almacén.

La segunda cuestión es cómo ejecutar el proceso de refresco. La primera opción es realizar una recarga completa de los datos operacionales, y la segunda opción es refrescar el almacén de forma incremental, es decir, detectar cambios de interés producidos en el sistema operacional desde el último refresco realizado, y propagar dichos cambios al almacén de datos. Generalmente es más utilizada la opción de refresco incremental que la de recarga completa.

La última se refiere a cuándo ejecutar el proceso de refresco. Algunos ejemplos posibles podrían ser: periódicamente (cada día a las 22:00, cada viernes a las 24:00, al final de cada mes, etc.), dependiendo de los cambios en los datos operacionales (por ejemplo cuando un atributo alcanza un valor límite, o bajo determinadas condiciones lógicas, etc.), y bajo petición expresa del administrador del almacén de datos.

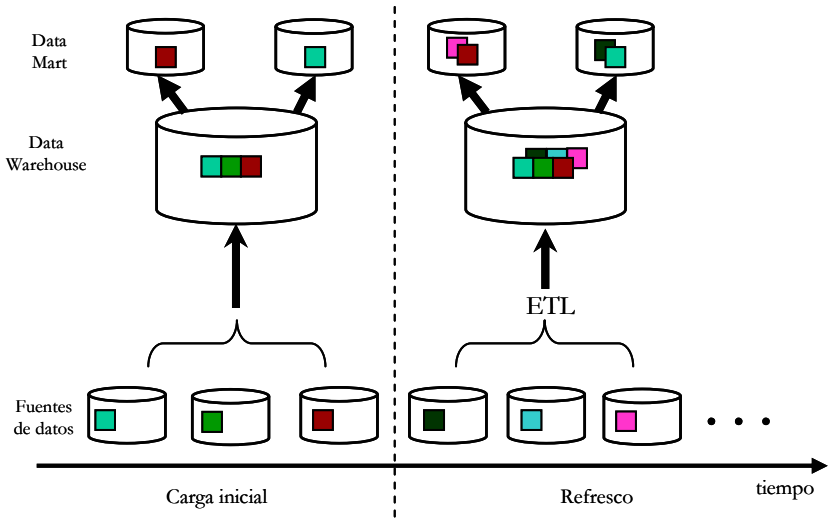

Figura 52. Carga inicial y refresco del almacén de datos

La Figura 52 ilustra las tareas relacionadas con el proceso de refresco. Después de la carga inicial del almacén, las actualizaciones de los datos operacionales se propaga hacia el almacén y los data marts en ciertos instantes de tiempo.

7.2.1 Arquitectura funcional

Como ya se dijo en el Capítulo 2, partiendo de la arquitectura para BD federadas definida en [Shet90], hemos definido una arquitectura integrada de BD que incorpora el concepto de almacén de datos y de data mart [Samo98]. En esta arquitectura, el almacén de datos y los data marts forman parte de la BD, son una funcionalidad más de la BD en lugar de ser sistemas independientes construidos utilizando una BD. A diferencia de otras propuestas, la nuestra permite que conceptos de BD federadas, temporales y orientadas a objetos, así

como de definición de esquemas externos, puedan ser aplicados directamente a la definición y construcción de almacenes de datos. Según la arquitectura de BD definida, el esquema de los almacenes de datos se define usando el MCD; sin embargo, los usuarios pueden trabajar sobre los almacenes de datos usando otros modelos de datos, según sus necesidades.

En la Figura 53 se muestra nuestra arquitectura integrada detallada considerando un solo esquema federado para simplificar la representación. El *esquema del almacén* y los *esquemas de data marts* se sitúan en paralelo con el resto de esquemas y pueden ser definidos a partir de cualquier esquema federado. En la figura también se representan los distintos procesadores utilizados para la conversión de esquemas entre niveles.

El almacén puede ser definido directamente tanto a partir del esquema federado como del ADO y, a diferencia del esquema federado, el almacén y los data marts se encuentran siempre materializados.

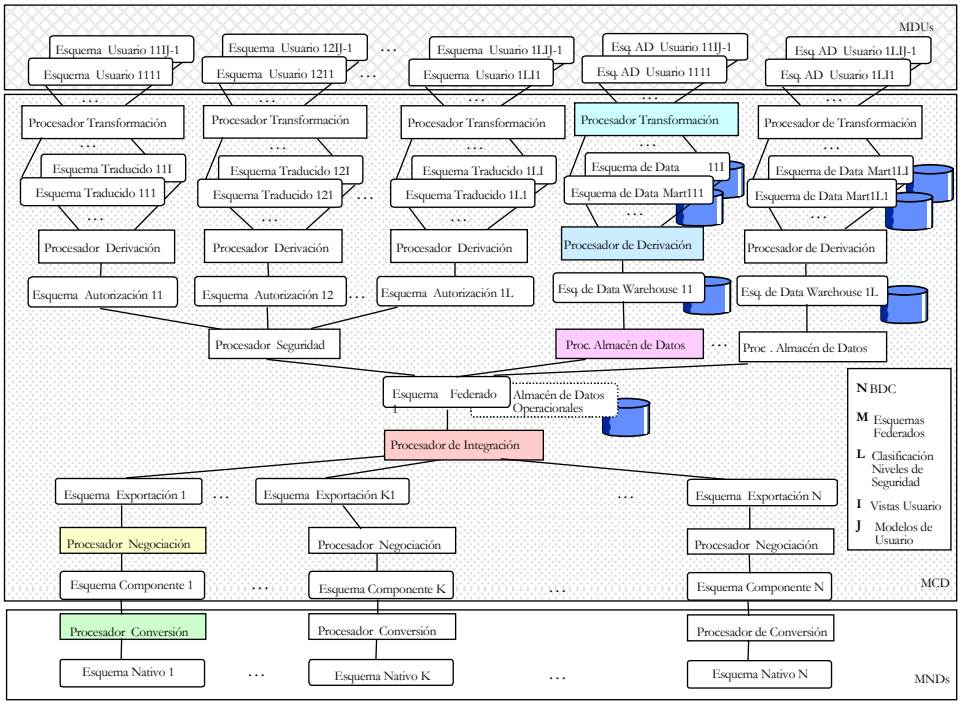

Figura 53. Arquitectura integrada de BD federadas y almacenes de datos detallada.

El *Esquema de Almacén de Datos* se define mediante el *Procesador de Almacén de Datos* que contempla las características de estructuración y almacenamiento de los datos del almacén de datos, así como también ofrece la posibilidad de considerar los distintos niveles de seguridad definidos. Los diferentes *Esquemas de Data Mart*, aunque fuera del ámbito de esta tesis, son derivados a partir del Esquema del almacén mediante un *Procesador de*

derivación que, al igual que el *Procesador de Almacén de Datos*, permite definir las características de estructuración y almacenamiento propias de los almacenes de datos.

Los esquemas del Almacén de Datos y de Data Marts son definidos usando el MCD de la federación; al igual que los Esquemas Traducidos, éstos pueden ser traducidos al modelo nativo de un usuario o grupo de usuarios federados mediante el *Procesador de Transformación*, obteniéndose así los *Esquemas de Almacén de Datos de Usuario.*

La arquitectura propuesta integra tanto el acceso en tiempo real como los almacenes de datos, el resultado es un sistema cooperativo para la integración de fuentes heterogéneas de información y almacenes de datos.

7.2.2 Arquitectura de operación/ejecución

Cuando la el Almacén de Datos ha sido construido, una arquitectura diferente, llamada *arquitectura de ejecución*, es la que entra en juego. La Figura 54 muestra esta arquitectura a un nivel *funcional*, representando todos los módulos funcionales y el flujo de datos desde el *inicio del refresco* a través de los módulos y los sistemas operacionales hasta obtener el *resultado del refresco.*

En la *fase de operación*, en primer lugar se lleva a cabo la descomposición de la consulta en subconsultas a las distintas fuentes de datos y la traducción de las subconsultas a la forma de los datos y al lenguaje de cada fuente (acciones realizadas por el *Gestor de Consultas* con la colaboración del *Gestor de Refresco*). El control de la ejecución de las subconsultas lo realiza el *Gestor de Transacciones*. Por último, la transformación de los subresultados, y combinación de los mismos para producir la respuesta consolidada al usuario es llevada a cabo de nuevo por el Gestor de Consultas [Rodr97].

El *Gestor de Transacciones* es el módulo responsable de la ejecución de las consultas. Utiliza el Árbol de Ejecución de la consulta para determinar a que fuente de datos se ha de enviar cada subconsulta y en qué orden. El resultado de cada subconsulta, si la ejecución global es correcta, se envía de nuevo al *Gestor de Consultas*. En caso de haberse producido algún error ha de realizar las acciones necesarias para evitar inconsistencias.

El *directorio* es el componente de la arquitectura funcional que se utiliza para almacenar toda la información (esquemas, correspondencias entre esquemas, …) que necesita el sistema para funcionar. El conjunto de información que se ha almacenar para el correcto funcionamiento del sistema consiste en:

- El almacén de metadatos de refresco.
- Las funciones de translación.
- Las correspondencias entre los diferentes esquemas.

7.2.2.1 El gestor de refresco

Es el módulo encargado de garantizar que las fuentes de datos son accedidas de manera coherente cada vez que se realiza un refresco de los datos del almacén. Para poder realizar este proceso el Gestor de Refresco se divide en dos submódulos: el *Controlador de Temporalidad* y el *Gestor de método de extracción*. A continuación se describe el funcionamiento de cada uno de ellos.

7.2.2.1.1 El Controlador de Temporalidad

El Controlador de Temporalidad es quien asegura que las fuentes de datos son accedidas en el momento adecuado. Se encarga de ejecutar los algoritmos de refresco descritos en este mismo capítulo para determinar la información que debe ser extraída de las fuentes de datos. Una vez determinadas las consultas que deben realizarse para realizar el refresco del almacén de datos estas se envían al Gestor de Consultas para que las transforme a consultas que puedan ser entendidas por las fuentes de datos.

7.2.2.1.2 El Gestor de Métodos de Extracción

Una vez transformadas las consultas al esquema nativo de las fuentes de datos, el Gestor de Métodos de Extracción se encarga de controlar que el conjunto de las transacciones se realizan de manera correcta, teniendo en cuenta el orden en el que deben realizarse y las dependencias entre ellas. Una vez determinada la forma en que deben ser ejecutadas las consultas a las fuentes de datos se envían al Gestor de Transacciones, donde se vigila que se realice el proceso de forma correcta.

7.2.2.2 El Gestor de Consultas

El Gestor de Consultas es el módulo encargado de transformar las consultas entre los diferentes esquemas de datos del sistema. Se puede dividir en cinco diferentes submódulos: el *Transformador de Consultas, el Descomponedor de Consultas, el Traductor de Consultas, el Gestor de Transformación y el Generador de Consultas para la Carga de Datos.*

Los tres primeros módulos forman parte de una primera fase, donde las consultas globales a los datos del esquema del almacén acaban transformándose a consultas en el formato nativo de las fuentes de datos, independientes entre sí.

En la segunda fase la transformación de la consultas se realiza en orden inverso, obteniendo una consulta global para introducir los datos obtenidos de las fuentes en el almacén. El primero de los módulos de esta segunda fase, denominado Gestor de Transformación, se encarga de integrar semánticamente los datos obtenidos de las fuentes de datos, tanto temporalmente (determinación de la granularidad de la información en el almacén,…) como en lo que respecta a la integración de los datos propiamente dicha (transformar un conjunto de valores de un parámetro semánticamente equivalente en un

solo valor). El segundo de los módulos genera las consultas necesarias para incluir esta información integrada en el Gestor de Transformación en el almacén de datos. Una vez obtenidas estas consultas, el Procesador de Refresco es el encargado de comprobar que se realizan de forma correcta.

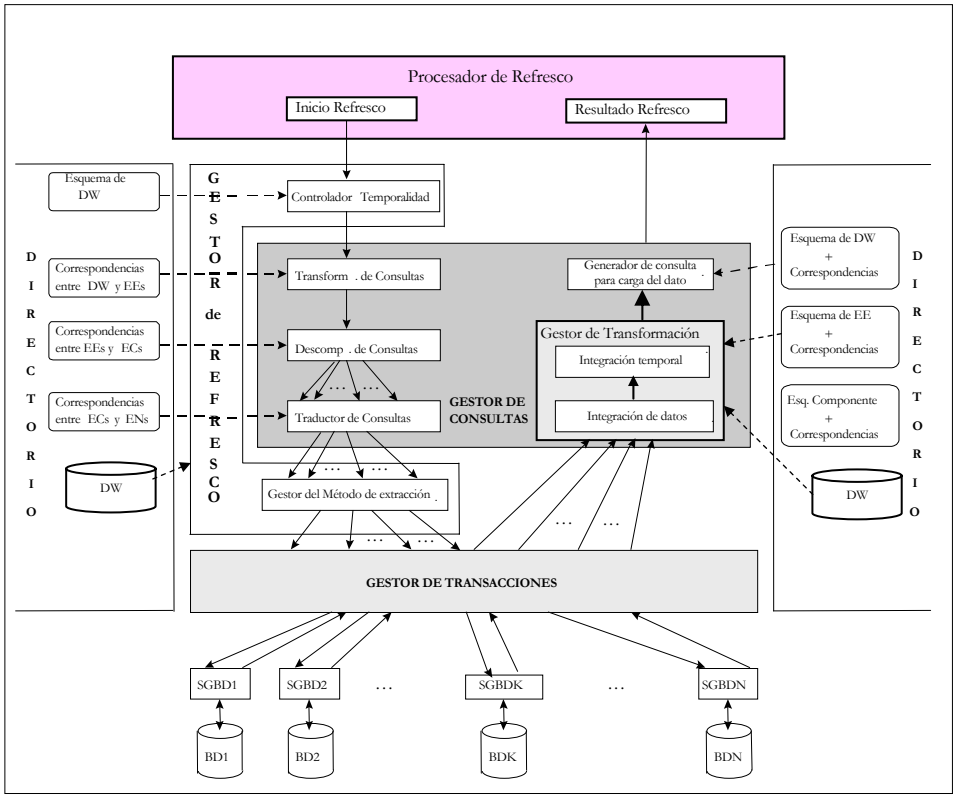

Figura 54. Arquitectura de operación/ejecución

7.3 Algoritmos de refresco

Como ejemplo para mostrar la utilidad algunos de los algoritmos que se describen a continuación se utiliza una aplicación desarrollada para los vuelos realizados por pilotos de vuelo libre. Estos pilotos dependen en gran medida de las condiciones meteorológicas para desarrollar su actividad y, una parte importante del sistema se encarga de manejar esta información. Se utilizan tres fuentes de datos para obtener este tipo de información:

- La página web del servicio nacional de meteorología de EEUU. Podemos acceder a mediciones de la temperatura, humedad, condiciones generales, presión y dirección y velocidad del viento realizadas cada media hora en cada uno de los aeropuertos del mundo. Sólo se refieren a la capa en contacto con la superficie de la atmósfera. Es una fuente de datos de tipo "file comparision".

- Para obtener un análisis más detallado y poder seleccionar la mejor zona de vuelo los pilotos utilizan otra herramienta, los diagramas de curvas de estado (SkewT diagrams). Estos diagramas de curvas de estado o sondeos son una captura de la temperatura, temperatura de condensación y el viento que hay a diferentes alturas sobre un mismo punto de la superficie de la tierra. Estos diagramas permiten obtener información acerca de la base y altura de las nubes, de la niebla, de las posibles inversiones térmicas que dificultan el ascenso de las masas de aire,… Estos sondeos se realizan en algunos aeropuertos cada doce horas lanzando un globo sonda. Aunque accesible mediante una interfaz web, esta fuente de datos esta organizada en forma de fichero log.
- A través de la página web del instituto nacional de meteorología se puede acceder a un número mayor de estaciones meteorológicas, pero únicamente a nivel nacional.

La información que proveen estas fuentes de datos en algunos casos es semánticamente equivalente. Dado un aeropuerto en el que se realizan sondeos, la información meteorológica de la capa superficial obtenida mediante un sondeo y la que se obtiene mediante las estaciones meteorológicas tradicionales debe ser igual si se refieren al mismo instante y mismo lugar.

Sucede lo mismo con algunas otras fuentes de datos del sistema. Podemos obtener información personal acerca de los pilotos que realizan los vuelos de distintas fuentes de datos:

- La base de datos propia, desarrollada para controlar el acceso de los usuarios a la plataforma web, contiene información personal para el propio funcionamiento de ésta (nombre, federación a la que pertenece, nivel de pilotaje,…). Sobre esta fuente de datos tenemos cierta flexibilidad a la hora de decidir el método de extracción que se va a utilizar. Se puede modificar la aplicación para que genere un fichero delta y almacene los cambios producidos en él temporalmente para permitir a otras aplicaciones tener acceso al mismo.
- Esta información también es accesible a través de la página web de las federaciones autonómicas de vuelo. Sobre esta última fuente de datos no tenemos ningún tipo de control, por lo que el único método de extracción de cambias que se le puede aplicar es el que se basa en la comparación de imágenes (File Comparision).
- También es posible acceder a la información sobre los pilotos de los clubes de vuelo. Si tenemos acceso a la aplicación con la que gestionan sus socios, podemos modificarla de forma que genere un fichero delta donde se almacenen temporalmente los cambios.

Para realizar la integración de estos datos es necesario utilizar el algoritmo correspondiente de entre los que se describen a continuación.

7.3.1 Método Application Assisted con el resto de métodos

7.3.1.1 AA – AA

La idea de este algoritmo es mantener una marca de los cambios que aún quedan por detectar en ambas fuentes. Cada cierto tiempo se ejecuta el algoritmo y se recorren las dos fuentes de datos a partir de esta marca temporal, integrando los pares de cambios. Se obtiene el primer cambio en la fuente 1 del parámetro que se desea integrar que se ha haya producido después de la marca que indica el primer cambio que no se ha podido integrar. Se realiza la misma operación en la segunda fuente de datos. Si alguno de estos dos cambios se ha producido después de la última vez que se realizó un refresco significa que es un cambio recogido por primera vez en alguna fuente y se puede llevar a cabo la integración. Al tratarse de ficheros delta deben aparecer todos los cambios repetidos en ambas fuentes y, además, ordenados temporalmente.

```
accesible = verdad
Si alguna fuente de datos no es periodica
        accesible = ComprobarVentana(Delta1)
        accesible = ComprobarVentana(Delta2) & accesible
Si accesible = verdad
        Repetir
                valorDelta1 = primerCambioDespuesDe(actualizadoAFecha1, Delta1)
                valorDelta2 = primerCambioDespuesDe(actualizadoAFecha2, Delta2)
                Si (TT(valorDelta1) > Mi-1 || TT(valorDelta2) > Mi-1) &  valorDelta1 <> null
                & valorDelta2 <> null
                        //El cambio es nuevo en alguna fuente
                        resultado = integrar(valorDelta1, valorDelta2)
                        actualizadoAFecha1 = TT(valorDelta1)
                        actualizadoAFecha2 = TT(valorDelta2)
        mientras valorDelta1 <> null && valorDelta2 <> null
```

En la Figura 55 se muestra un ejemplo de integración de los cambios producidos en dos fuentes de datos de tipo delta. En concreto, podría tratarse de dos de las fuentes de datos que se utilizan para obtener información acerca de los pilotos, la aplicación de gestión de algún club de vuelo y la aplicación desarrollada para el envío de vuelos a través de Internet. Supongamos que estamos tratando de integrar el nivel de pilotaje de un piloto en concreto. La cuarta vez que se consultan las fuentes de datos (instante M_4) no es posible integrar el cambio "L2" porque aún no esta disponible en una de las fuentes. La quinta vez que se consultan las fuentes se empieza a leer de nuevo a partir de las marcas temporales. En este caso, en la segunda fuente de datos si que esta registrado el cambio "L2", por lo que sabemos que es un cambio que no se había integrado antes. Se procede a su integración semántica y se realiza otra iteración del bucle principal del algoritmo. Al

detectar el cambio "L3" en ambas fuentes se puede proceder a integrar directamente, quedando las maracas temporales de ambas fuentes de datos apuntando a este último cambio. Este ejemplo es aplicable a cualquiera de los algoritmos donde se integren fuentes de tipo log entré sí, con ficheros delta o ficheros delta únicamente.

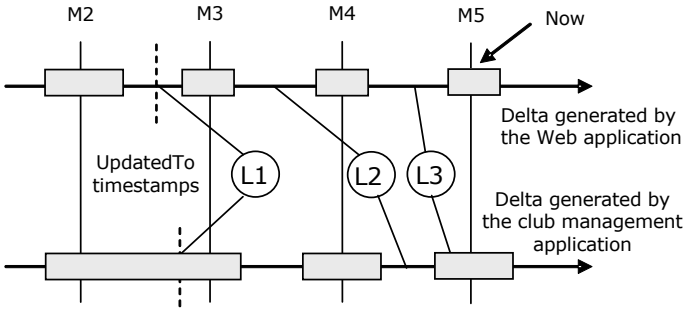

Figura 55. AA - AA

7.3.1.2 AA – TS

Cada vez que se accede a la fuente de datos del tipo TS se extrae el valor del parámetro que hay que integrar y se compara con el último valor conocido del mismo. Si se ha producido un cambio hay que buscar en la fuente de tipo delta el cambio asociado para realizar la integración. Puesto que la fuente de tipo delta puede haber recogido más de un cambio en el periodo que ha transcurrido desde el último refresco, únicamente se tiene en cuenta el último que se haya producido en este periodo. Esto se comprueba consultando el valor de TT del cambio en cuestión. Si se ha podido llevar a cabo la integración se actualiza el valor de la variable que almacena el valor anterior. Si no se ha podido realizar la integración no se actualiza el valor de esta variable, de forma que, en caso de detectarse el cambio en la fuente de tipo delta en refrescos posteriores se pueda llevar a cabo la integración incluso si no ha cambiado de nuevo el valor del parámetro en la fuente de tipo TS.

```
accesible = verdad
Si alguna fuente de datos no es periodica
        accesible = ComprobarVentana(AA)
        accesible = ComprobarVentana(TS) & accesible
Si accesible = verdad
        valorActual^TS = leerValor(TS)
        Si valorActual^TS <> valorAntiguo^TS
                valorActual^AA = último valor introducido en el delta para este parámetro
                Si TT(valorActual^AA) < M_i-1
                        No se puede integrar el valor porque el cambio detectado en TS  no
                        se ha introducido aún en el delta
                Si-no
                        resultado = Integrar(valorActual^TS, valorActual^AA)
                        // media, mayor, menor… de los dos
                        valorAntiguo^TS = valorActual^TS
```

El ejemplo de la figura 56 es similar al explicado en la sección que corresponde a la integración entre fuentes de tipo AA y FC.

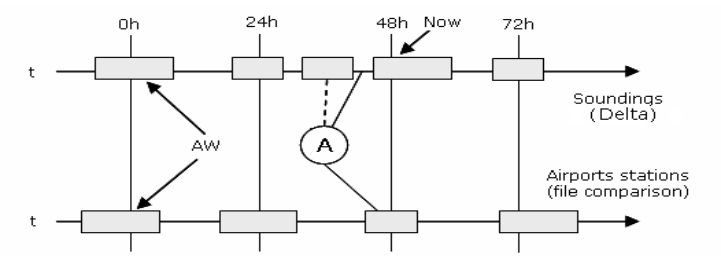

Figura 56. AA - Timestamp

7.3.1.3 AA – TΔ

La idea de este algoritmo es mantener una marca de los cambios que aún quedan por detectar en ambas fuentes. Cada cierto tiempo se ejecuta el algoritmo y se recorren las dos fuentes de datos a partir de esta marca temporal, integrando los pares de cambios. Se obtiene el primer cambio en la fuente 1 del parámetro que se desea integrar que se ha haya producido después de la marca que indica el primer cambio que no se ha podido integrar. Se realiza la misma operación en la segunda fuente de datos. Si alguno de estos dos cambios se ha producido después de la última vez que se realizó un refresco significa que es un cambio recogido por primera vez en alguna fuente y se puede llevar a cabo la integración. Al tratarse de ficheros delta deben aparecer todos los cambios repetidos en ambas fuentes y, además, ordenados temporalmente.

```
accesible = verdad
Si alguna fuente de datos no es periodica
        accesible = ComprobarVentana(Delta1)
        accesible = ComprobarVentana(Delta2) & accesible
Si accesible = verdad
        Repetir
                valorDelta1 = primerCambioDespuesDe(actualizadoAFecha1, Delta1)
                valorDelta2 = primerCambioDespuesDe(actualizadoAFecha2, Delta2)
                Si (TT(valorDelta1) > Mi-1 || TT(valorDelta2) > Mi-1) &  valorDelta1 <> null
                & valorDelta2 <> null
                        //El cambio es nuevo en alguna fuente
                        resultado = integrar(valorDelta1, valorDelta2)
                        actualizadoAFecha1 = TT(valorDelta1)
                        actualizadoAFecha2 = TT(valorDelta2)
        mientras valorDelta1 <> null && valorDelta2 <> null
```

El ejemplo de la figura 57 es similar al explicado en la sección que corresponde a la integración entre dos fuentes de tipo AA.

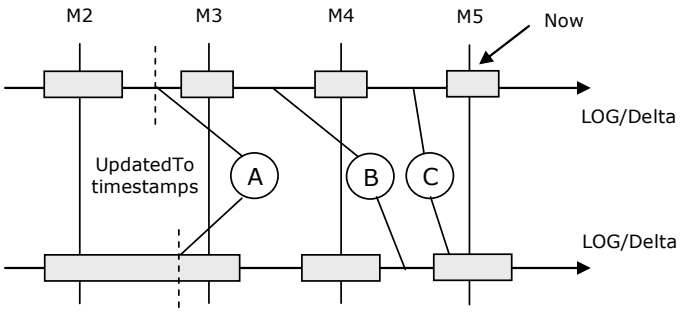

Figura 57. AA - TΔ

7.3.1.4 AA – TD

Se ejecutaría cada vez que se produce un cambio en la fuente TD. Sólo se procede a integrar el cambio si ha transcurrido un intervalo superior al valor del periodo de refresco. Ambas fuentes detectan todos los cambios producidos, pero puede ocurrir que cuando el trigger informe que se ha producido un cambio, éste aun no se haya introducido en el fichero delta. Ya que no es posible almacenar temporalmente el cambio (se trataría entonces de una fuente de datos de tipo triggered delta), se opta por descartar el cambio. Debido a estos descartes, en algunas ocasiones se da la situación de tener diferentes cambios en el delta, que aún no han sido integrados, que se pueden asociar al cambio que ha generado el trigger. Se ha optado por integrar el cambio del delta que más próximo se encuentre al instante en el que se generó el trigger que aún no haya sido integrado previamente, esto es, el más reciente.

```
accesible = verdad
Si alguna fuente de datos no es periodica
        accesible = ComprobarVentana(Delta)
        accesible = ComprobarVentana(TD) & accesible
Si accesible = verdad & (Ahora – fechaUltimoRefresco > M)
        cambio = UltimoCambio(Delta)
        si cambio <> null & TT(cambio) > actualizadoAFecha
                resultado = integrar(valorDelta, valorTD)
                actualizadoAFecha = Ahora
        fechaUltimoRefresco = Ahora
```

En la figura 58 se puede ver un ejemplo de la integración de dos fuentes de estos tipos. Podría tratarse de la misma situación planteada como ejemplo en el caso de la integración de dos fuentes de tipo AA, salvo que en este caso hemos optado por utilizar el método TD para extraer los cambios en el nivel de pilotaje de los pilotos de la aplicación que hemos desarrollado para el envío de vuelos a través de Internet. El último cambio integrado correctamente es L1, por lo que se coloca una marca temporal. El siguiente cambio que se produce, el cambio L2, lanza un trigger pero al no haber transcurrido un periodo de tiempo superior al valor del periodo de refresco no se procede a su integración. Cuando se lanza el trigger correspondiente al cambio L3 hay dos cambios en

la fuente de datos de tipo delta aún no han sido integrados. Se opta por integrar el cambio más reciente. Este ejemplo es aplicable a todas aquellas situaciones en las que se pretenden integrar fuentes de que generan ficheros de tipo log o delta con una fuente de tipo TD.

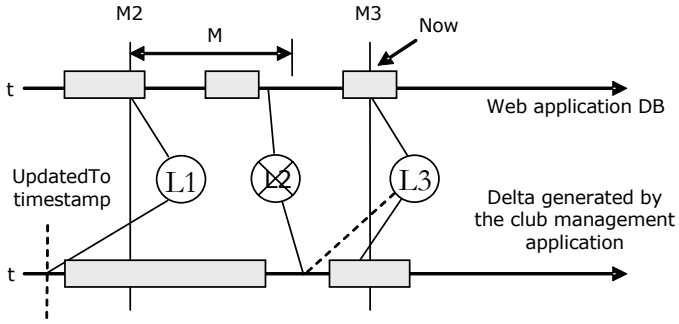

Figura 58. AA - TD

7.3.1.5 AA – FC

Cada vez que se accede a la fuente de datos del tipo FC se extrae el valor del parámetro que hay que integrar y se compara con el último valor conocido del mismo. Si se ha producido un cambio hay que buscar en la fuente de tipo delta el cambio asociado para realizar la integración. Puesto que la fuente de tipo delta puede haber recogido más de un cambio en el periodo que ha transcurrido desde el último refresco, únicamente se tiene en cuenta el último que se haya producido en este periodo. Esto se comprueba consultando el valor de TT del cambio en cuestión. Si se ha podido llevar a cabo la integración se actualiza el valor de la variable que almacena el valor anterior. Si no se ha podido realizar la integración no se actualiza el valor de esta variable, de forma que, en caso de detectarse el cambio en la fuente de tipo delta en refrescos posteriores se pueda llevar a cabo la integración incluso si no ha cambiado de nuevo el valor del parámetro en la fuente de tipo FC.

```
accesible = verdad
Si alguna fuente de datos no es periodica
        accesible = ComprobarVentana(AA)
        accesible = ComprobarVentana(FC) & accesible
Si accesible = verdad
        valorActual^Fc = leerValor(FC)
        Si valorActual^Fc <> valorAntiguo^Fc
                valorActual^AA = último valor introducido en el delta para este parámetro
                Si TT(valorActual^AA) < M_i-1
                        No se puede integrar el valor porque el cambio detectado en FC no
                        se ha introducido aún en el delta
                Si-no
                        resultado = Integrar(valorActual^Fc, valorActual^AA)
                        // media, mayor, menor… de los dos
                        valorAntiguo^Fc = valorActual^Fc
                        Eliminar fichero delta
```

Suponiendo que la figura 59 representa la evolución de las fuentes de datos meteorológicas del ejemplo que estamos siguiendo y que el diseñador desea obtener esta información con un nivel de detalle diario, el proceso de integración del cambio "A" detectado en la temperatura se realizaría de la siguiente forma: cada veinticuatro horas se consultan ambas fuentes; si el valor de la temperatura en la página web del aeropuerto ha cambiado con respecto al último que teníamos almacenado se recuperan los dos cambios del mismo parámetro que se han producido en la fuente correspondiente a los sondeos en las últimas veinticuatro horas (puesto que se realizan cada doce horas y se registran todos los cambios) y se realiza la integración semántica del valor de la página web con el último de ellos. Este ejemplo es aplicable a cualquiera en la que se desee realizar la integración entre una fuente de datos de tipo log o delta con otra fuente de tipo FC o TS.

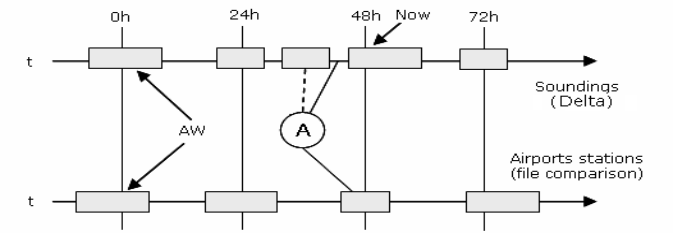

Figura 59. AA – FC

7.3.1.6 AA – LOG

La idea de este algoritmo es mantener una marca de los cambios que aún quedan por detectar en ambas fuentes. Cada cierto tiempo se ejecuta el algoritmo y se recorren las dos fuentes de datos a partir de esta marca temporal, integrando los pares de cambios. Se obtiene el primer cambio en la fuente 1 del parámetro que se desea integrar que se ha haya producido después de la marca que indica el primer cambio que no se ha podido integrar. Se realiza la misma operación en la segunda fuente de datos. Si alguno de estos dos cambios se ha producido después de la última vez que se realizó un refresco significa que es un cambio recogido por primera vez en alguna fuente y se puede llevar a cabo la integración. Al tratarse de ficheros delta y log deben aparecer todos los cambios repetidos en ambas fuentes y, además, ordenados temporalmente.

```
accesible = verdad
Si alguna fuente de datos no es periodica
        accesible = ComprobarVentana(Log)
        accesible = ComprobarVentana(Delta) & accesible
Si accesible = verdad
        Repetir
                valorDelta = primerCambioDespuesDe(actualizadoAFechaDelta, Delta)
                valorLog = primerCambioDespuesDe(actualizadoAFechaLog, Log)
                Si (TT(valorDelta) > Mi-1 || TT(valorLog) > Mi-1) &  valorDelta <> null &
                valorLog <> null
                        //El cambio es nuevo en alguna fuente
```

```
                    resultado = integrar(valorDelta, valorLog)
                    actualizadoAFechaDelta = TT(valorDelta)
                    actualizadoAFechaLog = TT(valorLog)
mientras valorDelta <> null && valorLog <> null
```

El ejemplo de la figura 60 es similar al explicado en la sección que corresponde a la integración entre dos fuentes de tipo AA.

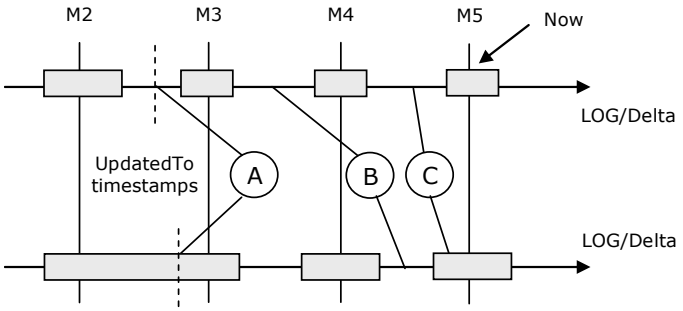

Figura 60. AA - Log

7.3.2 Método Timestamp con el resto de métodos

7.3.2.1 TS – TS

Cada cierto tiempo se comprueba el valor de ambas fuentes de datos. Si se detecta el cambio en ambas fuentes se procede a su integración.

```
accesible = verdad
Si alguna fuente de datos no es periodica
        accesible = ComprobarVentana(TS1)
        accesible = ComprobarVentana(TS2) & accesible
Si accesible = verdad
        valorActual^TS1 = leerValor(TS1)
        valorActual^TS2 = leerValor(TS2)
        Si (valorActual^TS1 <> valorAntiguo^TS1) & (valorActual^TS2 <> valorAntiguo^TS2)
                resultado = Integrar(valorActual^TS1, valorActual^TS2)
                valorAntiguo^TS1 = valorActual^TS1
                valorAntiguo^TS2 = valorActual^TS2
```

Se trata de la misma situación que se obtiene al intentar integrar dos fuentes de datos de tipo FC (figura 61).

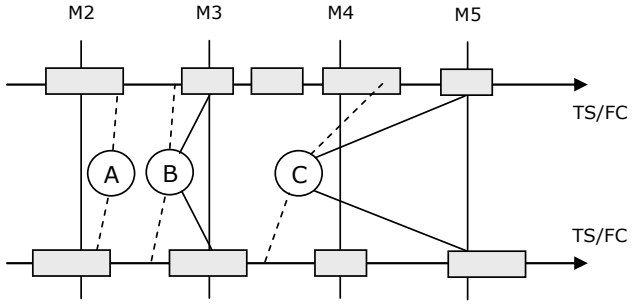

Figura 61. TS – TS

7.3.2.2 TS – TΔ

Cada vez que se accede a la fuente de datos del tipo TS se extrae el valor del parámetro que hay que integrar y se compara con el último valor conocido del mismo. Si se ha producido un cambio hay que buscar en la fuente de tipo delta el cambio asociado para realizar la integración. Puesto que la fuente de tipo delta puede haber recogido más de un cambio en el periodo que ha transcurrido desde el último refresco, únicamente se tiene en cuenta el último que se haya producido en este periodo. Esto se comprueba consultando el valor de TT del cambio en cuestión. Si se ha podido llevar a cabo la integración se actualiza el valor de la variable que almacena el valor anterior. Si no se ha podido realizar la integración no se actualiza el valor de esta variable, de forma que, en caso de detectarse el cambio en la fuente de tipo delta en refrescos posteriores se pueda llevar a cabo la integración incluso si no ha cambiado de nuevo el valor del parámetro en la fuente de tipo TS.

```
accesible = verdad
Si alguna fuente de datos no es periodica
        accesible = ComprobarVentana(TΔ)
        accesible = ComprobarVentana(TS) & accesible
Si accesible = verdad
        valorActual^TS = leerValor(TS)
        Si valorActual^TS <> valorAntiguo^TS
                valorActual^TΔ = último valor introducido en el delta para este parámetro
                Si TT(valorActual^TΔ) < M_i-1
                        No se puede integrar el valor porque el cambio detectado en TS no
                        se ha introducido aún en el delta
                Si-no
                        resultado = Integrar(valorActual^TS, valorActual^TΔ)
                        // media, mayor, menor… de los dos
                        valorAntiguo^TS = valorActual^TS
                        Eliminar fichero delta
```

El ejemplo de la figura 62 es similar al explicado en la sección que corresponde a la integración entre fuentes de tipo AA y FC.

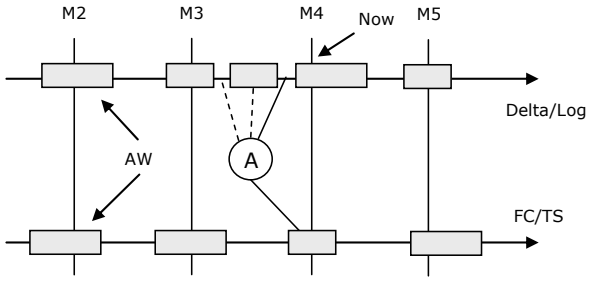

Figura 62. TS - TΔ

7.3.2.3 TS – TD

Se ejecutaría cada vez que se produce un cambio en la fuente TD. Sólo se procede a integrar el cambio si ha transcurrido un intervalo superior al valor del periodo de refresco. Cada vez que se genera un trigger se comprueba si se ha introducido el cambio en la fuente TS. Si es así, se integran. Si no, se ignora el cambio.

```
accesible = verdad
Si alguna fuente de datos no es periodica
        accesible = ComprobarVentana(TD)
        accesible = ComprobarVentana(TS) & accesible
Si accesible = verdad & (Ahora – fechaUltimoRefresco > M)
        valorActual^TS = leerValor(TS)
        Si valorActual^TS <> valorAntiguo^TS
                valorActual^TD = valor del cambio que ha generado el trigger
                resultado = Integrar(valorActual^TS, valorActual^TD)
                // media, mayor, menor… de los dos
                valorAntiguo^TS = valorActual^TS
                fechaUltimoRefresco = Ahora
```

Este algoritmo se utiliza en la aplicación de ejemplo para integrar los cambios producidos en las fuentes correspondientes a la información sobre los pilotos si optamos por utilizar el método de extracción trigger directo. Cuando un usuario de la plataforma web modifica algún parámetro de su información personal se lanza un trigger en la base de datos, que comprueba si ha pasado el tiempo suficiente que el administrador ha establecido entre procesos de refresco consecutivos. Si es así, se comprueba que en la página web de la federación también se haya producido un cambio en el mismo parámetro. En caso afirmativo se procede a la integración de estos dos valores. Sería el caso de los cambios A y C de la figura 63. Por el contrario, el cambio B no puede integrarse debido a que no ha transcurrido un intervalo de tiempo superior al valor del periodo de refresco. Es importante darse cuenta de que los cambios recogidos por la fuente de tipo TS son detectados en el momento en el que se realiza la comparación con el último valor que se tenía guardado, lo que implica que el cambio C de la figura es en realidad el cambio B. El proceso es similar entre fuentes de datos de tipo TD y FC, salvo que en las fuentes de datos de tipo TS se puede obtener el instante real en el que se produce el cambio.

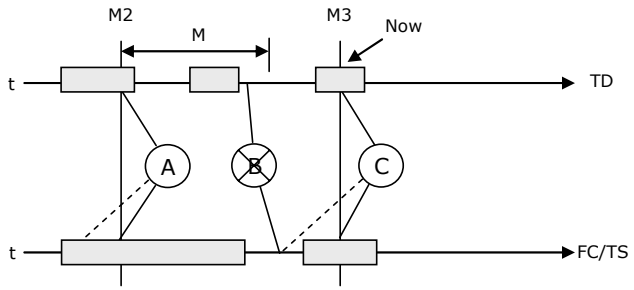

Figura 63. TS - TD

7.3.2.4 TS – FC

Cada cierto tiempo se comprueba el valor de ambas fuentes de datos. Si se detecta el cambio en ambas fuentes se procede a su integración. Se trata de la misma situación que se obtiene al intentar integrar dos fuentes de datos de tipo FC (figura 64).

accesible = verdad
Si alguna fuente de datos no es periodica
    accesible = ComprobarVentana(TS)
    accesible = ComprobarVentana(FC) & accesible
Si accesible = verdad
    $valorActual^{TS}$ = leerValor(TS)
    $valorActual^{FC}$ = leerValor(FC)
    Si ($valorActual^{TS}$ <> $valorAntiguo^{TS}$) & ($valorActual^{FC}$ <> $valorAntiguo^{FC}$)
        resultado = Integrar($valorActual^{TS}$, $valorActual^{FC}$)
        $valorAntiguo^{TS}$ = $valorActual^{TS}$
        $valorAntiguo^{FC}$ = $valorActual^{FC}$

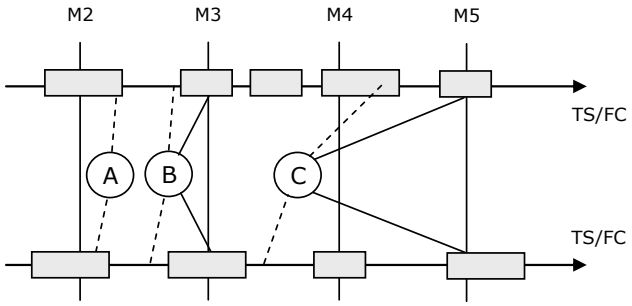

Figura 64. TS – FC

El ejemplo de la figura 62 es similar al explicado en la sección que corresponde a la integración entre dos fuentes de tipo FC.

7.3.2.5 TS – LOG

Cada vez que se accede a la fuente de datos del tipo TS se extrae el valor del parámetro que hay que integrar y se compara con el último valor conocido del mismo. Si se ha producido un cambio hay que buscar en la fuente de tipo delta el cambio asociado para realizar la integración. Puesto que la fuente de tipo log puede haber recogido más de un

cambio en el periodo que ha transcurrido desde el último refresco, únicamente se tiene en cuenta el último que se haya producido en este periodo. Esto se comprueba consultando el valor de TT del cambio en cuestión. Si se ha podido llevar a cabo la integración se actualiza el valor de la variable que almacena el valor anterior. Si no se ha podido realizar la integración no se actualiza el valor de esta variable, de forma que, en caso de detectarse el cambio en la fuente de tipo log en refrescos posteriores se pueda llevar a cabo la integración incluso si no ha cambiado de nuevo el valor del parámetro en la fuente de tipo TS.

```
accesible = verdad
Si alguna fuente de datos no es periodica
        accesible = ComprobarVentana(Log)
        accesible = ComprobarVentana(TS) & accesible
Si accesible = verdad
        valorActual^TS = leerValor(TS)
        Si valorActual^TS <> valorAntiguo^TS
                valorActual^Log = último valor introducido en el log para este parámetro
                Si TT(valorActual^Log) < M_i-1
                        No se puede integrar el valor porque el cambio detectado en TS no
                        se ha introducido aún en el log
                Si-no
                        resultado = Integrar(valorActual^TS, valorActual^Log)
                        // media, mayor, menor… de los dos
                        valorAntiguo^TS = valorActual^TS
```

El ejemplo de la figura 65 es similar al explicado en la sección que corresponde a la integración entre fuentes de tipo AA y FC.

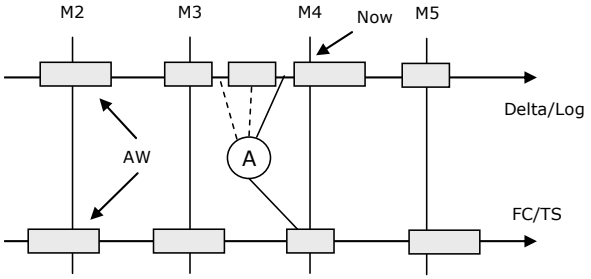

Figura 65. TS – Log

7.3.3 Método Trigger Delta con el resto de métodos

7.3.3.1 TΔ – TΔ

La idea de este algoritmo es mantener una marca de los cambios que aún quedan por detectar en ambas fuentes. Cada cierto tiempo se ejecuta el algoritmo y se recorren las dos fuentes de datos a partir de esta marca temporal, integrando los pares de cambios. Se

obtiene el primer cambio en la fuente 1 del parámetro que se desea integrar que se ha haya producido después de la marca que indica el primer cambio que no se ha podido integrar. Se realiza la misma operación en la segunda fuente de datos. Si alguno de estos dos cambios se ha producido después de la última vez que se realizó un refresco significa que es un cambio recogido por primera vez en alguna fuente y se puede llevar a cabo la integración. Al tratarse de ficheros delta deben aparecer todos los cambios repetidos en ambas fuentes y, además, ordenados temporalmente.

```
accesible = verdad
Si alguna fuente de datos no es periodica
        accesible = ComprobarVentana(Delta1)
        accesible = ComprobarVentana(Delta2) & accesible
Si accesible = verdad
        Repetir
                valorDelta1 = primerCambioDespuesDe(actualizadoAFecha1, Delta1)
                valorDelta2 = primerCambioDespuesDe(actualizadoAFecha2, Delta2)
                Si (TT(valorDelta1) > Mi-1 || TT(valorDelta2) > Mi-1) &  valorDelta1 <> null
                & valorDelta2 <> null
                        //El cambio es nuevo en alguna fuente
                        resultado = integrar(valorDelta1, valorDelta2)
                        actualizadoAFecha1 = TT(valorDelta1)
                        actualizadoAFecha2 = TT(valorDelta2)
        mientras valorDelta1 <> null && valorDelta2 <> null
```

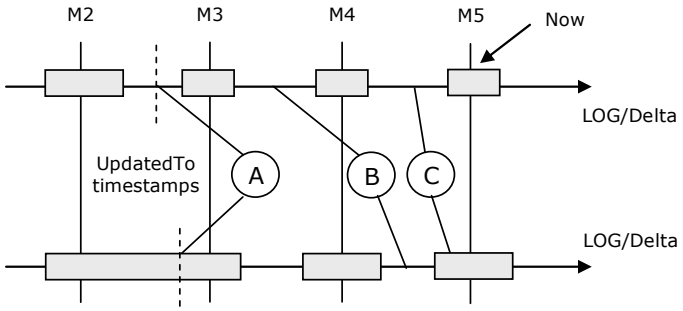

Figura 66. TΔ – TΔ

El ejemplo de la figura 66 es similar al explicado en la sección que corresponde a la integración entre dos fuentes de tipo AA.

7.3.3.2 TΔ – TD

Se ejecutaría cada vez que se produce un cambio en la fuente TD. Sólo se procede a integrar el cambio si ha transcurrido un intervalo superior al valor del periodo de refresco. Ambas fuentes detectan todos los cambios producidos, pero puede ocurrir que cuando el trigger informe que se ha producido un cambio, éste aun no se haya introducido en el fichero delta. Ya que no es posible almacenar temporalmente el cambio (se trataría entonces de una fuente de datos de tipo triggered delta), se opta por descartar el cambio. Debido a estos descartes, en algunas ocasiones se da la situación de tener diferentes

cambios en el delta, que aún no han sido integrados, que se pueden asociar al cambio que ha generado el trigger. Se ha optado por integrar el cambio del delta que más próximo se encuentre al instante en el que se generó el trigger que aún no haya sido integrado previamente, esto es, el más reciente.

```
accesible = verdad
Si alguna fuente de datos no es periodica
        accesible = ComprobarVentana(Delta)
        accesible = ComprobarVentana(TD) & accesible
Si accesible = verdad & (Ahora – fechaUltimoRefresco > M)
        cambio = UltimoCambio(Delta)
        si cambio <> null & TT(cambio) > actualizadoAFecha
                resultado = integrar(valorDelta, valorTD)
                actualizadoAFecha = Ahora
        fechaUltimoRefresco = Ahora
```

Se trata de la misma situación que se da a la hora de integrar temporalmente las fuentes de tipo AA con fuentes de tipo TD (figura 67).

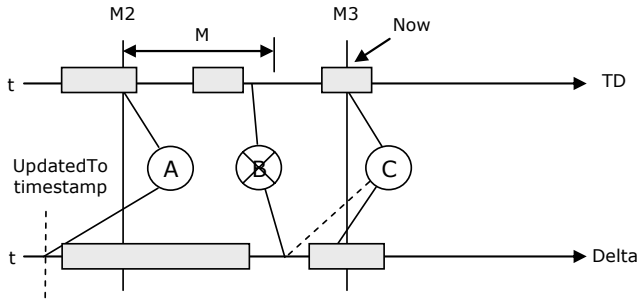

Figura 67. TΔ – TD

7.3.3.3 TΔ – FC

Cada vez que se accede a la fuente de datos del tipo FC se extrae el valor del parámetro que hay que integrar y se compara con el último valor conocido del mismo. Si se ha producido un cambio hay que buscar en la fuente de tipo delta el cambio asociado para realizar la integración. Puesto que la fuente de tipo delta puede haber recogido más de un cambio en el periodo que ha transcurrido desde el último refresco, únicamente se tiene en cuenta el último que se haya producido en este periodo. Esto se comprueba consultando el valor de TT del cambio en cuestión. Si se ha podido llevar a cabo la integración se actualiza el valor de la variable que almacena el valor anterior. Si no se ha podido realizar la integración no se actualiza el valor de esta variable, de forma que, en caso de detectarse el cambio en la fuente de tipo delta en refrescos posteriores se pueda llevar a cabo la integración incluso si no ha cambiado de nuevo el valor del parámetro en la fuente de tipo FC.

```
accesible = verdad
Si alguna fuente de datos no es periodica
```

```
        accesible = ComprobarVentana(AA)
        accesible = ComprobarVentana(FC) & accesible
Si accesible = verdad
        valorActual^Fc = leerValor(FC)
        Si valorActual^Fc <> valorAntiguo^Fc
                valorActual^AA = último valor introducido en el delta para este parámetro
                Si TT(valorActual^AA) < M_i-1
                        No se puede integrar el valor porque el cambio detectado en FC no
                        se ha introducido aún en el delta
                Si-no
                        resultado = Integrar(valorActual^Fc, valorActual^AA)
                        // media, mayor, menor… de los dos
                        valorAntiguo^Fc = valorActual^Fc
                        Eliminar fichero delta
```

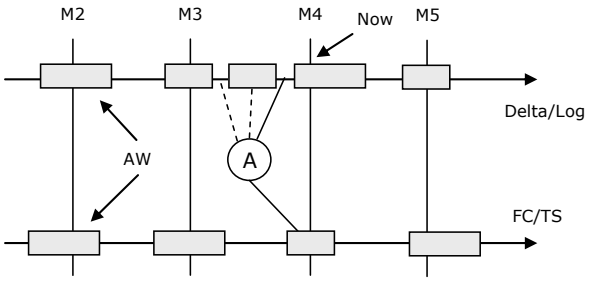

Figura 68. TΔ – FC

El ejemplo de la figura 68 es similar al explicado en la sección que corresponde a la integración entre fuentes de tipo AA y FC.

7.3.3.4 TΔ – LOG

La idea de este algoritmo es mantener una marca de los cambios que aún quedan por detectar en ambas fuentes. Cada cierto tiempo se ejecuta el algoritmo y se recorren las dos fuentes de datos a partir de esta marca temporal, integrando los pares de cambios. Se obtiene el primer cambio en la fuente 1 del parámetro que se desea integrar que se ha haya producido después de la marca que indica el primer cambio que no se ha podido integrar. Se realiza la misma operación en la segunda fuente de datos. Si alguno de estos dos cambios se ha producido después de la última vez que se realizó un refresco significa que es un cambio recogido por primera vez en alguna fuente y se puede llevar a cabo la integración. Al tratarse de ficheros delta y log deben aparecer todos los cambios repetidos en ambas fuentes y, además, ordenados temporalmente.

```
accesible = verdad
Si alguna fuente de datos no es periodica
        accesible = ComprobarVentana(Log)
        accesible = ComprobarVentana(Delta) & accesible
Si accesible = verdad
        Repetir
```

```
            valorDelta = primerCambioDespuesDe(actualizadoAFechaDelta, Delta)
            valorLog = primerCambioDespuesDe(actualizadoAFechaLog, Log)
            Si (TT(valorDelta) > Mi-1 || TT(valorLog) > Mi-1) &  valorDelta <> null &
            valorLog <> null
                        //El cambio es nuevo en alguna fuente
                        resultado = integrar(valorDelta, valorLog)
                        actualizadoAFechaDelta = TT(valorDelta)
                        actualizadoAFechaLog = TT(valorLog)
      mientras valorDelta <> null && valorLog <> null
```

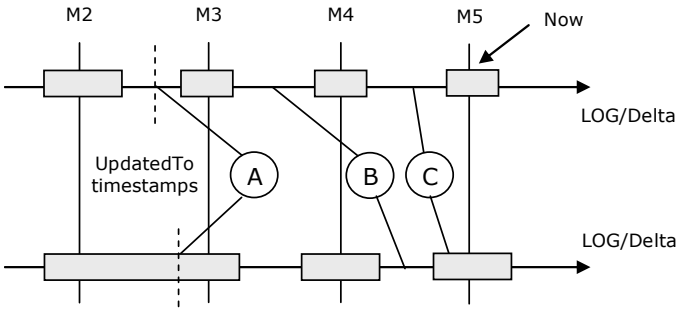

Figura 69. TΔ – Log

El ejemplo de la figura 69 es similar al explicado en la sección que corresponde a la integración entre dos fuentes de tipo AA.

7.3.4 Método Trigger Directo con el resto de métodos

7.3.4.1 TD – TD

No se pueden integrar dos fuentes de este tipo porque la integración debe llevarse a cabo cuando se lanzan los triggers y es prácticamente imposible que coincidan en el tiempo.

7.3.4.2 TD – FC

Se ejecutaría cada vez que se produce un cambio en la fuente TD. Sólo se procede a integrar el cambio si ha transcurrido un intervalo superior al valor del periodo de refresco. Cada vez que se genera un trigger se comprueba si se ha introducido el cambio en la fuente FC. Si es así, se integran. Si no, se ignora el cambio.

```
accesible = verdad
Si alguna fuente de datos no es periodica
        accesible = ComprobarVentana(TD)
        accesible = ComprobarVentana(FC) & accesible
Si accesible = verdad & (Ahora – fechaUltimoRefresco > M)
        valorActualFC = leerValor(FC)
        Si valorActualFC <> valorAntiguoFC
                valorActualTD = valor del cambio que ha generado el trigger
                resultado = Integrar(valorActualFC, valorActualTD)
```

// media, mayor, menor… de los dos
$valorAntiguo^{FC}$ = $valorActual^{FC}$
fechaUltimoRefresco = Ahora

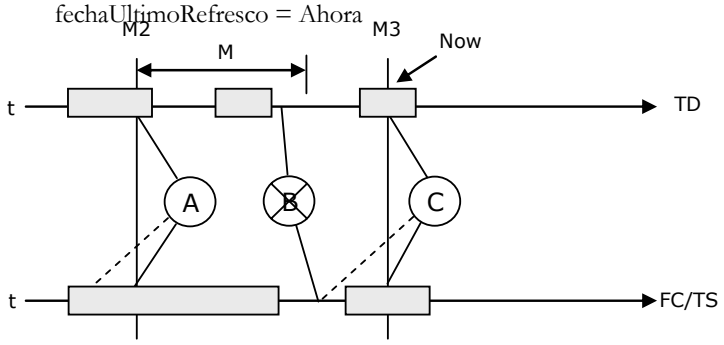

Figura 70. TD – FC

Se trata de la misma situación recogida en el caso de la integración de fuentes de datos de tipo TD y TS (figura 70).

7.3.4.3 TD – LOG

Se ejecutaría cada vez que se produce un cambio en la fuente TD. Sólo se procede a integrar el cambio si ha transcurrido un intervalo superior al valor del periodo de refresco. Ambas fuentes detectan todos los cambios producidos, pero puede ocurrir que cuando el trigger informe que se ha producido un cambio, éste aun no se haya introducido en el fichero log. Ya que no es posible almacenar temporalmente el cambio (se trataría entonces de una fuente de datos de tipo triggered delta), se opta por descartar el cambio. Debido a estos descartes, en algunas ocasiones se da la situación de tener diferentes cambios en el log, que aún no han sido integrados, que se pueden asociar al cambio que ha generado el trigger. Se ha optado por integrar el cambio del log que más próximo se encuentre al instante en el que se generó el trigger que aún no haya sido integrado previamente, esto es, el más reciente.

```
accesible = verdad
Si alguna fuente de datos no es periodica
        accesible = ComprobarVentana(Log)
        accesible = ComprobarVentana(TD) & accesible
Si accesible = verdad & (Ahora – fechaUltimoRefresco > M)
        cambio = UltimoCambio(Log)
        si cambio <> null & TT(cambio) > actualizadoAFecha
                resultado = integrar(valorLog, valorTD)
                actualizadoAFecha = Ahora
        fechaUltimoRefresco = Ahora
```

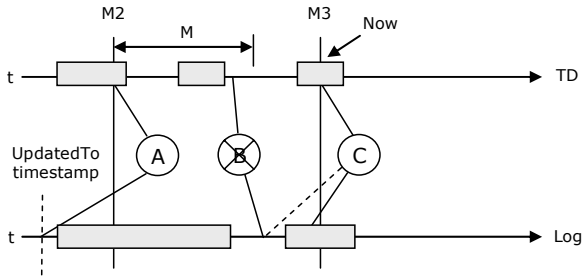

Figura 71. TD – Log

Se trata de la misma situación que se da a la hora de integrar temporalmente las fuentes de tipo AA con fuentes de tipo TD (figura 71).

7.3.5 Método File Comparision con el resto de métodos

7.3.5.1 FC – FC

Cada cierto tiempo se comprueba el valor de ambas fuentes de datos. Si se detecta el cambio en ambas fuentes se procede a su integración.

accesible = verdad
Si alguna fuente de datos no es periodica
    accesible = ComprobarVentana(FC1)
    accesible = ComprobarVentana(FC2) & accesible
Si accesible = verdad
    $valorActual^{Fc1}$ = leerValor(FC1)
    $valorActual^{Fc2}$ = leerValor(FC2)
    Si ($valorActual^{Fc1}$ <> $valorAntiguo^{Fc1}$) & ($valorActual^{Fc2}$ <> $valorAntiguo^{Fc2}$)
        resultado = Integrar($valorActual^{Fc1}$, $valorActual^{Fc2}$)
        $valorAntiguo^{Fc1}$ = $valorActual^{Fc1}$
        $valorAntiguo^{Fc2}$ = $valorActual^{Fc2}$

Este algoritmo es el que permite integrar las dos fuentes de datos, del ejemplo que estamos siguiendo, que permiten obtener información de las estaciones meteorológicas tradicionales. Ambas fuentes de datos son accesibles a través de páginas web y, por tanto, la forma de extraer los cambios que en ellas se producen es mediante el método de extracción FC. La única forma de integrar estos cambios es consultar cada cierto tiempo ambas fuentes. Si se ha producido un cambio con respecto al último valor conocido del parámetro a integrar en ambas fuentes se procede a su integración. El cambio B, recogido en la figura 72, se puede integrar en el instante M3, pues se ha detectado en ambas fuentes. No ocurre lo mismo con el cambio C de la misma figura, que debe esperar a instante M5 para que sea recogido por la primera de las fuentes de datos.

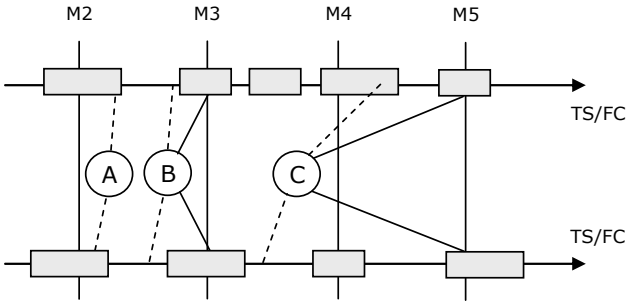

Figura 72. FC – FC

También puede suceder que no se detecte alguno de los cambios producidos (como el cambio A de la figura) porque su valor sea sobrescrito por otro cambio posterior (en este caso, el cambio B). Este mismo ejemplo es aplicable a la integración entre fuentes de tipo FC y TS, salvo que en las fuentes de datos de tipo TS si que es posible obtener el instante en el que se produjo el cambio.

7.3.5.2 FC – LOG

Cada vez que se accede a la fuente de datos del tipo FC se extrae el valor del parámetro que hay que integrar y se compara con el último valor conocido del mismo. Si se ha producido un cambio hay que buscar en la fuente de tipo log el cambio asociado para realizar la integración. Puesto que la fuente de tipo log puede haber recogido más de un cambio en el periodo que ha transcurrido desde el último refresco, únicamente se tiene en cuenta el último que se haya producido en este periodo. Esto se comprueba consultando el valor de TT del cambio en cuestión. Si se ha podido llevar a cabo la integración se actualiza el valor de la variable que almacena el valor anterior. Si no se ha podido realizar la integración no se actualiza el valor de esta variable, de forma que, en caso de detectarse el cambio en la fuente de tipo log en refrescos posteriores se pueda llevar a cabo la integración incluso si no ha cambiado de nuevo el valor del parámetro en la fuente de tipo FC.

```
accesible = verdad
Si alguna fuente de datos no es periodica
        accesible = ComprobarVentana(Log)
        accesible = ComprobarVentana(FC) & accesible
Si accesible = verdad
        valorActual^FC = leerValor(FC)
        Si valorActual^FC <> valorAntiguo^FC
                valorActual^Log = último valor introducido en el log para este parámetro
                Si TT(valorActual^Log) < M_i-1
                        No se puede integrar el valor porque el cambio detectado en FC no
                        se ha introducido aún en el log
                Si-no
                        resultado = Integrar(valorActual^FC, valorActual^Log)
                        // media, mayor, menor… de los dos
```

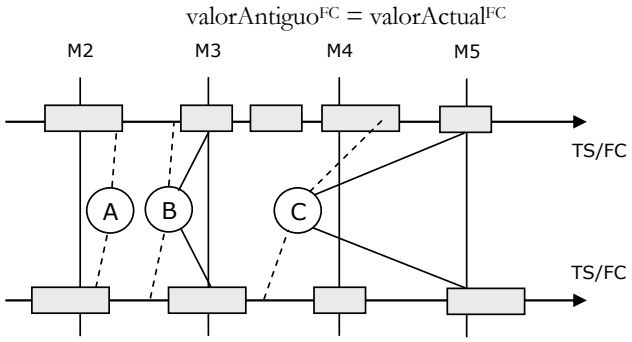

Figura 73. FC – Log

El ejemplo de la figura 73 es similar al explicado en la sección que corresponde a la integración entre fuentes de tipo AA y FC.

7.3.6 Método LOG con el resto de métodos

7.3.6.1 LOG – LOG

La idea de este algoritmo es mantener una marca de los cambios que aún quedan por detectar en ambas fuentes. Cada cierto tiempo se ejecuta el algoritmo y se recorren las dos fuentes de datos a partir de esta marca temporal, integrando los pares de cambios. Se obtiene el primer cambio en la fuente 1 del parámetro que se desea integrar que se ha haya producido después de la marca que indica el primer cambio que no se ha podido integrar. Se realiza la misma operación en la segunda fuente de datos. Si alguno de estos dos cambios se ha producido después de la última vez que se realizó un refresco significa que es un cambio recogido por primera vez en alguna fuente y se puede llevar a cabo la integración. Al tratarse de ficheros log deben aparecer todos los cambios repetidos en ambas fuentes y, además, ordenados temporalmente (ver figura 74).

```
accesible = verdad
Si alguna fuente de datos no es periodica
        accesible = ComprobarVentana(Log1)
        accesible = ComprobarVentana(Log2) & accesible
Si accesible = verdad
        Repetir
                valorLog1 = primerCambioDespuesDe(actualizadoAFechaLog1, Log1)
                valorLog2 = primerCambioDespuesDe(actualizadoAFechaLog,2 Log2)
                Si (TT(valorLog1) > Mi-1 || TT(valorLog2) > Mi-1) &  valorLog1 <> null &
                valorLog2 <> null
                        //El cambio es nuevo en alguna fuente
                        resultado = integrar(valorLog1, valorLog2)
                        actualizadoAFechaLog1 = TT(valorLog1)
                        actualizadoAFechaLog2 = TT(valorLog2)
        mientras valorLog1 <> null && valorLog2 <> null
```

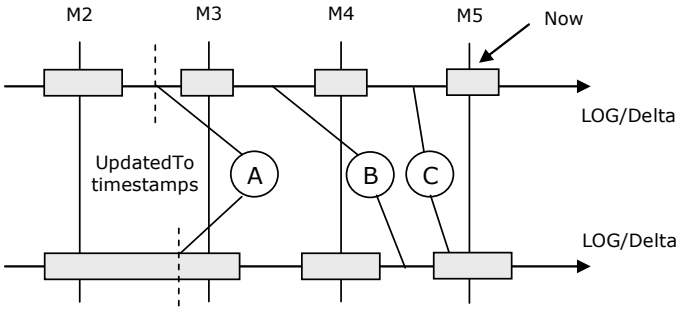

Figura 74. Log – Log

El ejemplo de la figura 73 es similar al explicado en la sección que corresponde a la integración entre fuentes de tipo AA.

7.4 Ejemplo ilustrativo

Para ilustrar el funcionamiento del sistema se describe a continuación un ejemplo de integración de un dato encontrado semánticamente equivalente en dos fuentes diferentes de la aplicación desarrollada para ayudar a la toma de decisiones de los pilotos de vuelo libre, previamente utilizada en esta tesis [Araq06]. En concreto se va a realizar la integración del parámetro "nivel", que se corresponde con la habilidad de los pilotos para aprovechar las características de la atmósfera. Se trata de seguir la arquitectura de la Figura 54.

```
accesible = verdad
Si alguna fuente de datos no es periodica
    accesible = ComprobarVentana(Log)
    accesible = ComprobarVentana(FC) & accesible
Si accesible = verdad
    valorActual^FC = leerValor(FC)
    Si valorActual^FC <> valorAntiguo^FC
        valorActual^Log = último valor introducido en el log
        para este parámetro
        Si TT(valorActual^Log) < M_i-1
            No se puede integrar el valor porque el cambio
            detectado en FC no se ha introducido aún en el
            log
        Si-no
            resultado = Integrar(valorActual^FC,
            valorActual^Log)
            // media, mayor, menor… de los dos
            valorAntiguoFC = valorActualFC
```

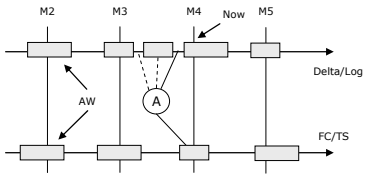

Figura 75. Algotitmo elegido

Cuando el *procesador de refresco* lo estima oportuno, dados los metadatos de refresco y los requisitos temporales proporcionados por el administrador del almacén, se lanza el *Controlador de Temporalidad* del *Gestor de Refresco.* Este módulo determina que para extraer la información de las fuentes de datos la información correspondiente al nivel del pilo es necesario consultar las fuentes de datos correspondientes a la federación autonómica de

parapente y a la base de datos de pilotos de la aplicación Web desarrollada para el envío de vuelos a través de Internet. Para realizar esta operación y por el tipo de fuentes de las que se trata el Controlador de Temporalidad determina que es necesario aplicar el algoritmo de extracción FC-LOG (Figura 75), recogido en este mismo capítulo. Este proceso requiere realizar una serie de consultas a las fuentes de datos, que deben ser traducidas al modelo de datos nativo de cada una de las fuentes. De esta transformación se encarga el *Gestor de Consultas*.

A partir de las funciones de translación que podemos ver en la Figura 76, el Transformador de Consultas obtiene los atributos de las clases del esquema externo de las fuentes de datos que deben integrarse. En este ejemplo concreto, para obtener el atributo nivel integrado en el almacén es necesario obtener el atributo Category de la clase `Instructor_T'` de la primera fuente de datos y el atributo `Level_T` de la clase `Confirmed'` de la segunda de las fuentes, como se muestra en la Figura 77. Se generan entonces las consultas necesarias para acceder a estos atributos del esquema externo de las fuentes de datos.

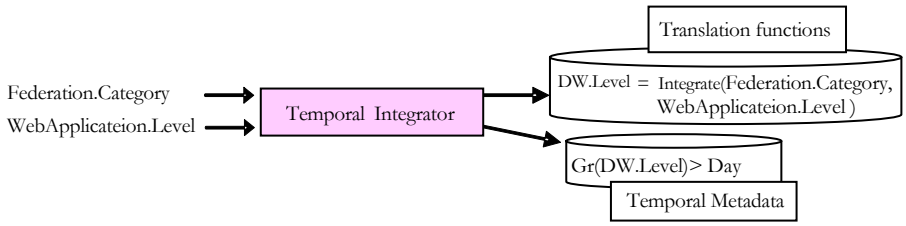

Figura 76. Funciones de Traslación y Metadatos

El esquema externo es en realidad una vista particular del esquema componente de las fuentes de datos, introduciendo clases derivadas que ocultan parte del esquema, por lo que hace falta traducir estas consultas. De esta tarea se encarga el *Descomponedor de Consultas*. En el ejemplo que estamos siguiendo, en lugar de realizar una consulta a las fuentes para extraer el atributo Category de la clase `Instructor_T'`, se transformaría de forma que se obtuviese el mismo atributo pero de la clase `Instructor_T`, como puede verse en la Figura 77.

El último paso que se debe realizar en el *Gestor de Consultas* antes de acceder a las fuentes de datos es traducir las consultas del modelo canónico de datos del sistema a un formato que pueda entender cada una de las fuentes de datos. Siguiendo con el ejemplo de la aplicación de los pilotos de vuelo libre, habría que cambiar el formato de las fechas para adaptarlas al formato con el que aparecen en las fuentes de datos (utilizar dos dígitos para el año en lugar de cuatro; mostrar sólo los días, meses y años, separados por barras de dividir,…).

Cuando ya se tiene traducida toda la información necesaria para realizar la consulta al esquema nativo de las fuentes de datos, el *Gestor del Método de Extracción* se encarga de

preparar las consultas para puedan ser ejecutadas directamente por el *Gestor de Transacciones*. Si se trata de una base de datos relacional habrá que generar una consulta en SQL, en cambio, si se debe acceder mediante un wrapper habrá que indicarle los parámetros necesarios para obtener la información deseada [Araq03] [Araq03b]. En el ejemplo que estamos siguiendo, este módulo determina primero el monitor correspondiente a la aplicación web y le instaría a determinar el valor actual del atributo `Level`. A continuación determina el wrapper correspondiente a la fuente de datos de la federación autonómica de parapente y le transfiere los parámetros necesarios para que obtenga el último cambio realizado en el fichero log para el parámetro `Category`. En realidad no se accede directamente a las fuentes de datos, sino que se introduce una capa que otorga cierto nivel de abstracción a los métodos de extracción de las fuentes de datos.

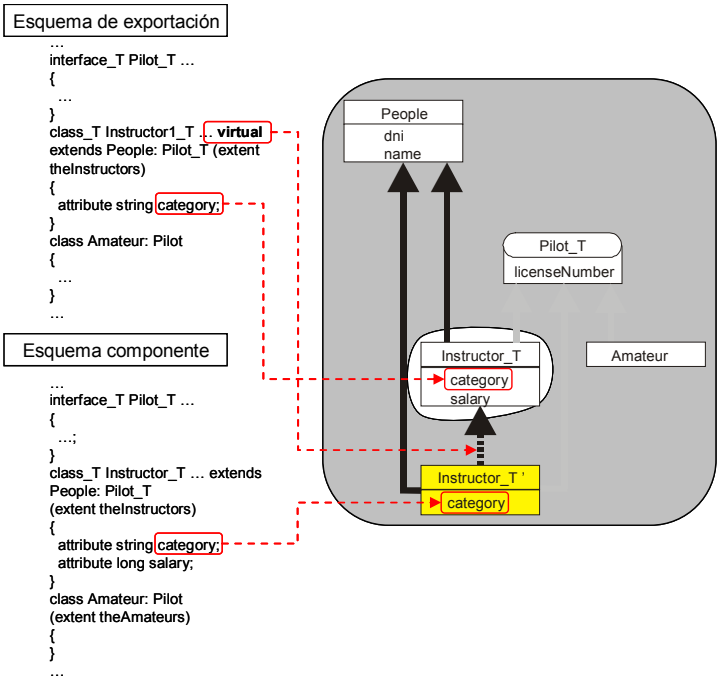

Figura 77. Equivalencias entre esquemas

Esta capa, correspondiente al *Gestor de Transacciones*, es la encargada de controlar que las consultas a las fuentes de datos se realicen correctamente, determinando el orden de las transacciones, esperando a que todas las que dependan entre sí devuelvan terminen, gestionando los posibles errores que puedan producirse, favorecer la ejecución en paralelo,…

A continuación se realiza la integración temporal del mismo valor, adaptándolo a los requisitos temporales que requiere el almacén para el atributo, que vienen descritos en el almacén de *metadatos temporales*. En el ejemplo que estamos siguiendo, la granularidad del valor integrado se establece a nivel de día, aunque en una de las fuentes de datos se haya determinado con un nivel de detalle de minuto.

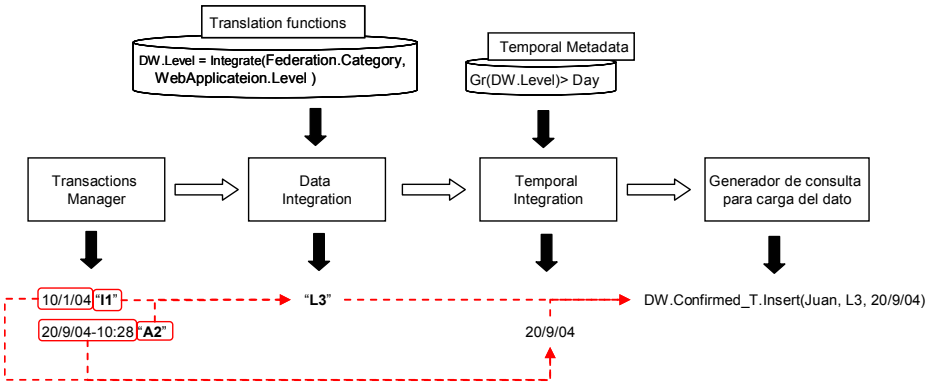

Figura 78. Proceso previo a la actualización del almacén

Por último, consultando de nuevo las correspondencias entre los esquemas de exportación y del almacén, se construye la consulta que realiza la inserción del dato en el almacén, integrado tanto a nivel de datos como temporalmente. El *Procesador de Refresco* es el encargado de ejecutar esta consulta. Todos estos pasos los podemos ver en la Figura 78.

7.5 Conclusiones

Podemos ver al almacén de datos como una base de datos que recopila y almacena información procedente de diferentes fuentes de datos o sistemas operacionales. Desde este punto de vista, el contenido del mismo es un conjunto de datos materializadas, basados en fuentes distribuidas, homogéneas y autónomas. Estas datos cambian de valor con el paso del tiempo y, dado que el almacén ofrece una visión histórica de los datos, es necesario actualizar el almacén con los cambios producidos en las fuentes.

En este capítulo se ha propuesto una arquitectura de operación para llevar a cabo el proceso de refresco del almacén de datos. Esta arquitectura se basa en varios módulos cada uno de los cuáles se encarga de realizar una función dentro del proceso. Se han propuesto los algoritmos necesarios para realizar el refresco teniendo en cuenta el método de extracción y la fuente en concreto a consultar.

Capítulo 8

CONCLUSIONES Y TRABAJO FUTURO

8.1 Conclusiones

Las principales contribuciones aportadas en esta tesis son:

- Enriquecimiento del modelo de objetos ODMG 3.0 con elementos temporales. ODMG carece de una representación diferenciada de elementos con semántica temporal. Partiendo de esta carencia hemos ampliado su modelo de objetos con los elementos temporales necesarios para la definición de los esquemas de las fuentes y del almacén de datos.
- Extensión de los metadatos de ODMG con el objetivo de poder incorporar los nuevos elementos definidos para el modelo de objetos. La extensión de los metadatos, permite, además de almacenar en el repositorio información sobre los esquemas de las fuentes y del almacén de datos, definir los metadatos de las fuentes que formarán parte del almacén de datos.
- Ampliación de la arquitectura propuesta por Sheth y Larson a una de 7 niveles en la que se incluyen, por una parte los esquemas de exportación con semántica temporal de las fuentes de datos que se van a integrar en el almacén y, por otra el propio esquema del almacén de datos. Tanto los esquemas de exportación como el esquema del almacén se definen utilizando la extensión del modelo de objetos ODMG desarrollada en esta tesis.
- Propuesta de una arquitectura funcional de refresco que será utilizada durante la operación del sistema. Esta arquitectura incorpora el gestor de refresco encargado de controlar los pasos necesarios para realizar la actualización del almacén.
- Ampliación de la metodología de integración de esquemas con el propósito de incluir el procesador de integración de propiedades temporales de los datos, y el generador de metadatos para el refresco. Estos módulos se encargan de elegir y ejecutar el algoritmo de integración adecuado y de generar los metadatos correspondientes.

- Desarrollo de algoritmos para la integración de las propiedades temporales de los datos. Estos algoritmos, teniendo en cuenta los datos y los métodos para extraerlos de las fuentes, nos indican si los datos se pueden integrar y, además, en caso de que la integración sea posible, el nivel de detalle con el que se puede llevar a cabo dicha integración. Se basan en las propiedades temporales de los datos y de los métodos de extracción. Para desarrollar los algoritmos previamente se ha realizado un estudio de los diferentes métodos de extracción y se han fijado los parámetros que se han considerado de interés.
- Desarrollo de algoritmos para la actualización del almacén de datos. Una vez que un dato se incorpora al almacén hay que actualizarlo según los requerimientos del administrador del almacén de datos. Estos algoritmos se encargan de refrescar los datos del almacén basándose en los métodos de extracción que se utilizan para la obtención del dato de la fuente.

8.2 Trabajo futuro

Las principales líneas de trabajo futuro son:

- Descripción de las fuentes de datos y de los métodos de extracción mediante el uso de ontologías, de forma que los datos estén anotados correctamente para poder realizar la integración de instancias.
- Enriquecimiento semántico de las fuentes con propiedades espacio-temporales. Para ello será necesario un estudio del dominio espacio-temporal en el que se encuentran definidas.
- Desarrollo de algoritmos para la integración de propiedades espacio-temporales de los datos. Una vez enriquecidas las fuentes y los métodos con los parámetros espacio-temporales adecuados serán necesarios los algoritmos para decidir si se pueden integrar los datos y en qué forma se puede llevar a cabo esta integración.

APENDICE A

A.1 Cruces entre métodos de extracción

En este apéndice se incluyen semi-formalizados los cruces entre los métodos basándose en sus parámetros temporales que han sido comentados en la sección 6.3.

AA – AA (AA = Application – Assisted :: FD1 = Fuente de Datos 1 :: FD2 = Fuente de Datos 2)

AA (FD1) → AA (FD2) ↓	T. de ajuste	T. mínimo de consulta	T. de extracción	Ventana disponibilidad	Periodo de muestro	TT	T. de almacenamiento del log, delta, fichero imagen	Disponi.. de cambios
T. de ajuste	SI; TA =Max (TA(FD1), TA(FD2)) ∧ TA < VDi							
T. mínimo de consulta		SI; TMC =Max (TMC(FD1), TMC(FD2))						
T. de extracción			SI; TE = (TE(FD1)∧TE(FD2))⊂ (VD(FD1)∩VD(FD2)) ó TE = (TEfin(FD1)∧TEfin(FD2))⊂ (VD(FD1)∩VD(FD2)) ó TE = max(TE(FD1), TE(FD2))					
Ventana disponibilidad				SI; VD = VD(FD1)∩VD(FD2)				
Periodo de muestro					SI; (M(FD1) ∧ M (FD2)) ⊂ (VD(FD1)∩VD(FD2))			
TT						SI; TT (FD1) = TT (FD2)		
T. de almacenamiento del log, delta, fichero imagen							SI; TA ⊂ (VD(FD1)∩VD(FD2))	
Disponibilidad de cambios								SI; Todos los incluidos en ambos ficheros delta.

AA – TS (AA = Application – Assisted :: TS = TimeStamp :: FD1 = Fuente de Datos 1 :: FD2 = Fuente de Datos 2)

AA (FD1) → / TS (FD2) ↓	T. de ajuste	T. mínimo de consulta	T. de extracción	Ventana disponibilidad	Periodo de muestro	TT	T. de almacenamiento del log, delta, fichero imagen	Disponi.. de cambios
T. de ajuste	SI; TA =Max (TA(FD1), TA(FD2)) ∧ TA < VDi							
T. mínimo de consulta		SI; TMC = TMC(AA)						
T. de extracción			SI; TE = (TE(FD1)∧TE(FD2))⊂ (VD(FD1)∩VD(FD2)) ó TE = (TEfin(FD1)∧TEfin(FD2))⊂ (VD(FD1)∩VD(FD2))					
Ventana disponibilidad				SI; VD = VD(FD1)∩VD(FD2)				
Periodo de muestro					SI; (M(FD1) ∧ M (FD2)) ⊂ (VD(FD1)∩VD(FD2))			
TT						SI; TT (FD1) = TT (FD2)		
T. de almacenamiento del log, delta, fichero imagen							SI; TA (AA) ⊂ M(TS)	
Disponibilidad de cambios								NO; Solo algunos, los disponibles en la fuente TS.

AA – TΔ (AA = Application – Assisted :: TΔ = Triggered Delta :: FD1 = Fuente de Datos 1 :: FD2 = Fuente de Datos 2)

AA (FD1) → TΔ (FD2) ↓	T. de ajuste	T. mínimo de consulta	T. de extracción	Ventana disponibilidad	Periodo de muestro	TT	T. de almacenamiento del log, delta, fichero imagen	Disponi.. de cambios
T. de ajuste	SI; TA =Max (TA(FD1), TA(FD2)) ∧ TA < VDi ó TA = TA(AA) ∧ TA < VDi							
T. mínimo de consulta		SI; TMC =Max (TMC(FD1), TMC(FD2)) ó TMC = TMC(AA)						
T. de extracción			SI; TE = (TE(FD1)∧TE(FD2))⊂ (VD(FD1)∩VD(FD2)) ó TE = (TEfin(FD1)∧TEfin(FD2))⊂ (VD(FD1)∩VD(FD2))					
Ventana disponibilidad				SI; VD = VD(FD1)∩VD(FD2)				
Periodo de muestro					SI; (M(FD1) ∧ M (FD2)) ⊂ (VD(FD1)∩VD(FD2))			
TT						SI; TT (FD1) = TT (FD2)		
T. de almacenamiento del log, delta, fichero imagen							SI; TA ⊂ (VD(FD1)∩VD(FD2))	
Disponibilidad de cambios								SI; Todos los incluidos en ambos ficheros delta.

AA – TD (AA = Application – Assisted :: TD = Triggered Direct :: FD1 = Fuente de Datos 1 :: FD2 = Fuente de Datos 2)

AA (FD1) → TD (FD2) ↓	T. de ajuste	T. mínimo de consulta	T. de extracción	Ventana disponibilidad	Periodo de muestro	TT	T. de almacenamiento del log, delta, fichero imagen	Disponi.. de cambios
T. de ajuste	SI; TA =Max (TA(FD1), TA(FD2)) ∧ TA < VDi ó TA = TA(AA) ∧ TA < VDi							
T. mínimo de consulta		SI; TMC =Max (TMC(FD1), TMC(FD2))						
T. de extracción			SI; TE = TE(AA)					
Ventana disponibilidad				SI (condicional); VD = VD (AA) Si TE(TD) ⊂ VD(AA) entonces "SI se puede hacer la integración" en caso contrario "NO se puede hacer la integración"				
Periodo de muestro					NO			
TT						SI (condicional); Si suponemos que el TT de TD se captura para cada dato y se almacena en algún lugar.		
T. de almacenamiento del log, delta, fichero imagen							NO	
Disponibilidad de cambios								SI (condicional); Tenemos todos los cambios con ambos métodos pero hay q ver si se puede hacer la integración

AA – FC (AA = Application – Assisted :: FC = File Comparison :: FD1 = Fuente de Datos 1 :: FD2 = Fuente de Datos 2)

AA (FD1) → / FC (FD2) ↓	T. de ajuste	T. mínimo de consulta	T. de extracción	Ventana disponibilidad	Periodo de muestro	TT	T. de almacenamiento del log, delta, fichero imagen	Disponi.. de cambios
T. de ajuste	SI; TA =Max (TA(FD1), TA(FD2)) ∧ TA < VDi							
T. mínimo de consulta		SI; TMC =Max (TMC(FD1), TMC(FD2))						
T. de extracción			SI; TE = TE(FC)					
Ventana disponibilidad				SI; VD = VD(FD1)∩VD(FD2)				
Periodo de muestro					SI; (M(FD1) ∧ M (FD2)) ⊂ (VD(FD1)∩VD(FD2))			
TT						NO		
T. de almacenamiento del log, delta, fichero imagen							SI; TA ⊂ (VD(FD1)∩VD(FD2))	
Disponibilidad de cambios								NO; Solo algunos, los disponibles en FC

AA – LOG (AA = Application – Assisted :: LOG = Log :: FD1 = Fuente de Datos 1 :: FD2 = Fuente de Datos 2)

AA (FD1) → / AA (FD2) ↓	T. de ajuste	T. mínimo de consulta	T. de extracción	Ventana disponibilidad	Periodo de muestro	TT	T. de almacenamiento del log, delta, fichero imagen	Disponi.. de cambios
T. de ajuste	SI; TA =Max (TA(FD1), TA(FD2)) ∧ TA < VDi							
T. mínimo de consulta		SI; TMC =Max (TMC(FD1), TMC(FD2))						
T. de extracción			SI; TE = (TE(FD1)∧TE(FD2))⊂ (VD(FD1)∩VD(FD2)) ó TE = (TEfin(FD1)∧TEfin(FD2))⊂ (VD(FD1)∩VD(FD2))					
Ventana disponibilidad				SI; VD = VD(FD1)∩VD(FD2)				
Periodo de muestro					SI; (M(FD1) ∧ M (FD2)) ⊂ (VD(FD1)∩VD(FD2))			
TT						SI; TT (FD1) = TT (FD2)		
T. de almacenamiento del log, delta, fichero imagen							SI; TA ⊂ (VD(FD1)∩VD(FD2))	
Disponibilidad de cambios								SI; Todos los incluidos en ambos ficheros delta.

TS – TS (TS = TimeStamp :: FD1 = Fuente de Datos 1 :: FD2 = Fuente de Datos 2)

TS (FD1) → / TS (FD2) ↓	T. de ajuste	T. mínimo de consulta	T. de extracción	Ventana disponibilidad	Periodo de muestro	TT	T. de almacenamiento del log, delta, fichero imagen	Disponi.. de cambios
T. de ajuste	SI; TA =Max (TA(FD1), TA(FD2)) ∧ TA < VDi							
T. mínimo de consulta		SI; TMC =Max (TMC(FD1), TMC(FD2))						
T. de extracción			SI; TE = (TE(FD1)∧TE(FD2))⊂ (VD(FD1)∩VD(FD2)) ó TE = (TEfin(FD1)∧TEfin(FD2))⊂ (VD(FD1)∩VD(FD2))					
Ventana disponibilidad				SI; VD = VD(FD1)∩VD(FD2)				
Periodo de muestro					SI; (M(FD1) ∧ M (FD2)) ⊂ (VD(FD1)∩VD(FD2))			
TT						SI; TT (FD1) = TT (FD2)		
T. de almacenamiento del log, delta, fichero imagen							SI; TA (AA) ⊂ M(TS) TA = min(TA(FD1),TA(FD2))	
Disponibilidad de cambios								NO; Solo algunos

TS – TΔ (TS = TimeStamp :: TΔ = Triggered Delta :: FD1 = Fuente de Datos 1 :: FD2 = Fuente de Datos 2)

TS (FD1) → TΔ (FD2) ↓	T. de ajuste	T. mínimo de consulta	T. de extracción	Ventana disponibilidad	Periodo de muestro	TT	T. de almacenamiento del log, delta, fichero imagen	Disponi.. de cambios
T. de ajuste	SI; TA =Max (TA(FD1), TA(FD2)) ∧ TA < VDi ó TA = TA(AA) ∧ TA < VDi							
T. mínimo de consulta		SI; TMC =Max (TMC(FD1), TMC(FD2)) ó TMC = TMC(AA)						
T. de extracción			SI; TE = (TE(FD1)∧TE(FD2))⊂ (VD(FD1)∩VD(FD2)) ó TE = (TEfin(FD1)∧TEfin(FD2))⊂ (VD(FD1)∩VD(FD2))					
Ventana disponibilidad				SI; VD = VD(FD1)∩VD(FD2)				
Periodo de muestro					SI; (M(FD1) ∧ M (FD2)) ⊂ (VD(FD1)∩VD(FD2))			
TT						SI; TT (FD1) = TT (FD2)		
T. de almacenamiento del log, delta, fichero imagen							SI; TA = TA(F2)	
Disponibilidad de cambios								No; Solo algunos. Los incluidos en TΔ

TS – TD (TS = Timestamp :: TD = Triggered Direct :: FD1 = Fuente de Datos 1 :: FD2 = Fuente de Datos 2)

TS (FD1) → / TD (FD2) ↓	T. de ajuste	T. mínimo de consulta	T. de extracción	Ventana disponibilidad	Periodo de muestro	TT	T. de almacenamiento del log, delta, fichero imagen	Disponi.. de cambios
T. de ajuste	SI; TA =Max (TA(FD1), TA(FD2)) ∧ TA < VDi ó TA = TA(AA) ∧ TA < VDi							
T. mínimo de consulta		SI; TMC =Max (TMC(FD1), TMC(FD2))						
T. de extracción			SI; TE = TE(FD1)					
Ventana disponibilidad				SI (condicional); VD = VD (TS) Si TE(TD) ⊂ VD(TS) entonces "SI se puede hacer la integración" en caso contrario "NO se puede hacer la integración"				
Periodo de muestro					NO			
TT						SI (condicional); Si suponemos que el TT de TD se captura para cada dato y se almacena en algún lugar.		
T. de almacenamiento del log, delta, fichero imagen							NO	
Disponibilidad de cambios								Algunos; Todos los de la fuente TD

TS – FC (TS = Timestamp:: FC = File Comparison :: FD1 = Fuente de Datos 1 :: FD2 = Fuente de Datos 2)

TS (FD1) → / FC (FD2) ↓	T. de ajuste	T. mínimo de consulta	T. de extracción	Ventana disponibilidad	Periodo de muestro	TT	T. de almacenamiento del log, delta, fichero imagen	Disponi.. de cambios
T. de ajuste	SI; TA =Max (TA(FD1), TA(FD2)) ∧ TA < VDi							
T. mínimo de consulta		SI; TMC =Max (TMC(FD1), TMC(FD2))						
T. de extracción			SI; TE = TE(FD2)					
Ventana disponibilidad				SI; VD = VD(FD1)∩VD(FD2)				
Periodo de muestro					SI; (M(FD1) ∧ M (FD2)) ⊂ (VD(FD1)∩VD(FD2))			
TT						NO		
T. de almacenamiento del log, delta, fichero imagen							NO	
Disponibilidad de cambios								NO; Solo algunos

TS – LOG (TS = Timestamp :: LOG = Log :: FD1 = Fuente de Datos 1 :: FD2 = Fuente de Datos 2)

TS (FD1) → / LOG (FD2) ↓	T. de ajuste	T. mínimo de consulta	T. de extracción	Ventana disponibilidad	Periodo de muestro	TT	T. de almacenamiento del log, delta, fichero imagen	Disponi.. de cambios
T. de ajuste	SI; TA =Max (TA(FD1), TA(FD2)) ∧ TA < VDi							
T. mínimo de consulta		SI; TMC =Max (TMC(FD1), TMC(FD2))						
T. de extracción			SI; TE = (TE(FD1)∧TE(FD2))⊂ (VD(FD1)∩VD(FD2)) ó TE = (TEfin(FD1)∧TEfin(FD2))⊂ (VD(FD1)∩VD(FD2))					
Ventana disponibilidad				SI; VD = VD(FD1)∩VD(FD2)				
Periodo de muestro					SI; (M(FD1) ∧ M (FD2)) ⊂ (VD(FD1)∩VD(FD2))			
TT						SI; TT (FD1) = TT (FD2)		
T. de almacenamiento del log, delta, fichero imagen							SI; TA = TA(FD2)	
Disponibilidad de cambios								NO; Solo algunos. Los recogidos en la fuente FD2.

TΔ – TΔ (TΔ = Triggered Delta :: FD1 = Fuente de Datos 1 :: FD2 = Fuente de Datos 2)

TΔ (FD1) → / TΔ (FD2) ↓	T. de ajuste	T. mínimo de consulta	T. de extracción	Ventana disponibilidad	Periodo de muestro	TT	T. de almacenamiento del log, delta, fichero imagen	Disponi.. de cambios
T. de ajuste	SI; TA =Max (TA(FD1), TA(FD2)) ∧ TA < VDi ó TA = TA(AA) ∧ TA < VDi							
T. mínimo de consulta		SI; TMC =Max (TMC(FD1), TMC(FD2)) ó TMC = TMC(AA)						
T. de extracción			SI; TE = (TE(FD1)∧TE(FD2))⊂ (VD(FD1)∩VD(FD2)) ó TE = (TEfin(FD1)∧TEfin(FD2))⊂ (VD(FD1)∩VD(FD2))					
Ventana disponibilidad				SI; VD = VD(FD1)∩VD(FD2)				
Periodo de muestro					SI; (M(FD1) ∧ M (FD2)) ⊂ (VD(FD1)∩VD(FD2))			
TT						SI; TT (FD1) = TT (FD2)		
T. de almacenamiento del log, delta, fichero imagen							SI; TA ⊂ (VD(FD1)∩VD(FD2)) TA = min(TA(FD1), TA(FD2))	
Disponibilidad de cambios								SI; Todos los incluidos en ambos ficheros delta.

TΔ – TD (TΔ = Triggered Delta :: TD = Triggered Direct :: FD1 = Fuente de Datos 1 :: FD2 = Fuente de Datos 2)

TΔ (FD1) → / TD (FD2) ↓	T. de ajuste	T. mínimo de consulta	T. de extracción	Ventana disponibilidad	Periodo de muestro	TT	T. de almacenamiento del log, delta, fichero imagen	Disponi.. de cambios
T. de ajuste	SI; TA =Max (TA(FD1), TA(FD2)) ∧ TA < VDi ó TA = TA(AA) ∧ TA < VDi							
T. mínimo de consulta		SI; TMC =Max (TMC(FD1), TMC(FD2))						
T. de extracción			SI; TE = TE(FD1)					
Ventana disponibilidad				SI (condicional); VD = VD (TΔ) Si TE(TD) ⊂ VD(TΔ) entonces "SI se puede hacer la integración" en caso contrario "NO se puede hacer la integración"				
Periodo de muestro					NO			
TT						SI (condicional); Si suponemos que el TT de TD se captura para cada dato y se almacena en algún lugar.		
T. de almacenamiento del log, delta, fichero imagen							NO	
Disponibilidad de cambios								SI (condicional); Tenemos todos los cambios con ambos métodos pero hay q ver si se puede hacer la integración

TΔ – FC (TΔ = Triggered Delta :: FC = File Comparison :: FD1 = Fuente de Datos 1 :: FD2 = Fuente de Datos 2)

TΔ (FD1) → FC (FD2) ↓	T. de ajuste	T. mínimo de consulta	T. de extracción	Ventana disponibilidad	Periodo de muestro	TT	T. de almacenamiento del log, delta, fichero imagen	Disponi.. de cambios
T. de ajuste	SI; TA =Max (TA(FD1), TA(FD2)) ∧ TA < VDi							
T. mínimo de consulta		SI; TMC =Max (TMC(FD1), TMC(FD2))						
T. de extracción			SI; TE = TE(FC)					
Ventana disponibilidad				SI; VD = VD(FD1)∩VD(FD2)				
Periodo de muestro					SI; (M(FD1) ∧ M (FD2)) ⊂ (VD(FD1)∩VD(FD2))			
TT						NO		
T. de almacenamiento del log, delta, fichero imagen							SI; TA ⊂ (VD(FD1)∩VD(FD2)) TA = TA(FD1)	
Disponibilidad de cambios								NO; Solo algunos, los disponibles en FC

TΔ – LOG (TΔ = Triggered Delta :: LOG = Log :: FD1 = Fuente de Datos 1 :: FD2 = Fuente de Datos 2)

TΔ (FD1) → AA (FD2) ↓	T. de ajuste	T. mínimo de consulta	T. de extracción	Ventana disponibilidad	Periodo de muestro	TT	T. de almacenamiento del log, delta, fichero imagen	Disponi.. de cambios
T. de ajuste	SI; TA =Max (TA(FD1), TA(FD2)) ∧ TA < VDi							
T. mínimo de consulta		SI; TMC =Max (TMC(FD1), TMC(FD2))						
T. de extracción			SI; TE = (TE(FD1)∧TE(FD2))⊂ (VD(FD1)∩VD(FD2)) ó TE = (TEfin(FD1)∧TEfin(FD2))⊂ (VD(FD1)∩VD(FD2))					
Ventana disponibilidad				SI; VD = VD(FD1)∩VD(FD2)				
Periodo de muestro					SI; (M(FD1) ∧ M (FD2)) ⊂ (VD(FD1)∩VD(FD2))			
TT						SI; TT (FD1) = TT (FD2)		
T. de almacenamiento del log, delta, fichero imagen							SI; TA ⊂ (VD(FD1)∩VD(FD2)) TA = min(TA(FD1), TA(FD2))	
Disponibilidad de cambios								SI; Todos los incluidos en ambos ficheros.

TD – TD (TD = Triggered Direct :: FD1 = Fuente de Datos 1 :: FD2 = Fuente de Datos 2)

TD (FD1) → TD (FD2) ↓	T. de ajuste	T. mínimo de consulta	T. de extracción	Ventana disponibilidad	Periodo de muestro	TT	T. de almacenamiento del log, delta, fichero imagen	Disponi.. de cambios
T. de ajuste	SI; TA =Max (TA(FD1), TA(FD2)) ∧ TA < VDi ó TA = TA(AA) ∧ TA < VDi							
T. mínimo de consulta		SI; TMC =Max (TMC(FD1), TMC(FD2))						
T. de extracción			NO					
Ventana disponibilidad				NO				
Periodo de muestro					NO			
TT						SI (condicional); Si suponemos que el TT de TD se captura para cada dato y se almacena en algún lugar.		
T. de almacenamiento del log, delta, fichero imagen							NO	
Disponibilidad de cambios								SI (condicional); Tenemos todos los cambios con ambos métodos pero hay q ver si se puede hacer la integración

TD – FC (TD = Triggered Direct :: FC = File Comparison :: FD1 = Fuente de Datos 1 :: FD2 = Fuente de Datos 2)

TD (FD1) → / FC (FD2) ↓	T. de ajuste	T. mínimo de consulta	T. de extracción	Ventana disponibilidad	Periodo de muestro	TT	T. de almacenamiento del log, delta, fichero imagen	Disponi.. de cambios
T. de ajuste	SI; TA =Max (TA(FD1), TA(FD2)) ∧ TA < VDi							
T. mínimo de consulta		SI; TMC =Max (TMC(FD1), TMC(FD2))						
T. de extracción			SI; TE = TE(FC)					
Ventana disponibilidad				SI; VD = VD(FD2)				
Periodo de muestro					NO			
TT						NO		
T. de almacenamiento del log, delta, fichero imagen							NO	
Disponibilidad de cambios								NO; Solo algunos, los disponibles en FC

TD – LOG (TD = Triggered Direct :: LOG = Log :: FD1 = Fuente de Datos 1 :: FD2 = Fuente de Datos 2)

TD (FD1) → AA (FD2) ↓	T. de ajuste	T. mínimo de consulta	T. de extracción	Ventana disponibilidad	Periodo de muestro	TT	T. de almacenamiento del log, delta, fichero imagen	Disponi.. de cambios
T. de ajuste	SI; TA =Max (TA(FD1), TA(FD2)) ∧ TA < VDi							
T. mínimo de consulta		SI; TMC =Max (TMC(FD1), TMC(FD2))						
T. de extracción			SI; TE = (TE(FD1)∧TE(FD2))⊂ (VD(FD1)∩VD(FD2)) ó TE = (TEfin(FD1)∧TEfin(FD2))⊂ (VD(FD1)∩VD(FD2))					
Ventana disponibilidad				SI; VD = VD(FD2)				
Periodo de muestro					NO			
TT						SI; TT (FD1) = TT (FD2)		
T. de almacenamiento del log, delta, fichero imagen							SI; TA = TA(FD2)	
Disponibilidad de cambios								SI (condicional); Tenemos todos los cambios con ambos métodos pero hay q ver si se puede hacer la integración

FC – FC (FC = File Comparison :: FD1 = Fuente de Datos 1 :: FD2 = Fuente de Datos 2)

FC (FD1) → / FC (FD2) ↓	T. de ajuste	T. mínimo de consulta	T. de extracción	Ventana disponibilidad	Periodo de muestro	TT	T. de almacenamiento del log, delta, fichero imagen	Disponi.. de cambios
T. de ajuste	SI; TA =Max (TA(FD1), TA(FD2)) ∧ TA < VDi							
T. mínimo de consulta		SI; TMC =Max (TMC(FD1), TMC(FD2))						
T. de extracción			SI; TE = (TE(FD1)∧TE(FD2))⊂ (VD(FD1)∩VD(FD2)) ó TE = (TEfin(FD1)∧TEfin(FD2))⊂ (VD(FD1)∩VD(FD2))					
Ventana disponibilidad				SI; VD = VD(FD1)∩VD(FD2)				
Periodo de muestro					SI; (M(FD1) ∧ M (FD2)) ⊂ (VD(FD1)∩VD(FD2))			
TT						NO		
T. de almacenamiento del log, delta, fichero imagen							NO	
Disponibilidad de cambios								NO; Solo algunos

FC – LOG (FC = File comparison :: LOG = Log :: FD1 = Fuente de Datos 1 :: FD2 = Fuente de Datos 2)

FC (FD1) → / LOG (FD2) ↓	T. de ajuste	T. mínimo de consulta	T. de extracción	Ventana disponibilidad	Periodo de muestro	TT	T. de almacenamiento del log, delta, fichero imagen	Disponi.. de cambios
T. de ajuste	SI; TA =Max (TA(FD1), TA(FD2)) ∧ TA < VDi							
T. mínimo de consulta		SI; TMC =Max (TMC(FD1), TMC(FD2))						
T. de extracción			SI; TE = (TE(FD1)∧TE(FD2))⊂ (VD(FD1)∩VD(FD2)) ó TE = (TEfin(FD1)∧TEfin(FD2))⊂ (VD(FD1)∩VD(FD2))					
Ventana disponibilidad				SI; VD = VD(FD1)∩VD(FD2)				
Periodo de muestro					SI; (M(FD1) ∧ M (FD2)) ⊂ (VD(FD1)∩VD(FD2))			
TT						NO		
T. de almacenamiento del log, delta, fichero imagen							SI; TA = TA(FD2)	
Disponibilidad de cambios								NO; Solo algunos. Los recogidos en la fuente FD2.

LOG – LOG (LOG = Log :: FD1 = Fuente de Datos 1 :: FD2 = Fuente de Datos 2)

LOG (FD1) → / LOG (FD2) ↓	T. de ajuste	T. mínimo de consulta	T. de extracción	Ventana disponibilidad	Periodo de muestro	TT	T. de almacenamiento del log, delta, fichero imagen	Disponi.. de cambios
T. de ajuste	SI; TA =Max (TA(FD1), TA(FD2)) ∧ TA < VDi							
T. mínimo de consulta		SI; TMC =Max (TMC(FD1), TMC(FD2))						
T. de extracción			SI; TE = (TE(FD1)∧TE(FD2))⊂ (VD(FD1)∩VD(FD2)) ó TE = (TEfin(FD1)∧TEfin(FD2))⊂ (VD(FD1)∩VD(FD2))					
Ventana disponibilidad				SI; VD = VD(FD1)∩VD(FD2)				
Periodo de muestro					SI; (M(FD1) ∧ M (FD2)) ⊂ (VD(FD1)∩VD(FD2))			
TT						SI; TT (FD1) = TT (FD2)		
T. de almacenamiento del log, delta, fichero imagen							SI; TA ⊂ (VD(FD1)∩VD(FD2)) TA = min(TA(FD1), TA(FD2))	
Disponibilidad de cambios								SI; Todos los incluidos en ambos ficheros.

APENDICE B

B.1 Herramienta software de ayuda

A continuación se describe una herramienta que permite tener cuenta todos los aspectos temporales que intervienen en la integración de diferentes fuentes de datos. Ha sido desarrollada siguiendo las ideas propuestas en esta tesis.

En la Figura 79 se puede ver una captura de pantalla de la aplicación. En la parte izquierda del área de trabajo se encuentra el esquema del almacénd e datos en forma de árbol. En principio, el esquema del almacén es generado de forma automática por el sistema a partir de las diferentes fuentes de datos, pero se le da al diseñador del almacén de datos la posibilidad de realizar modificaciones.

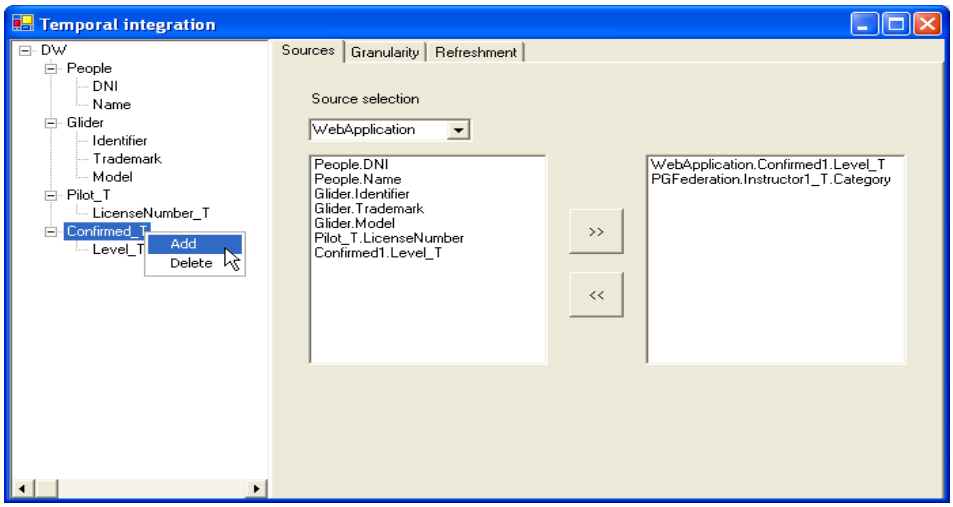

Figura 79. Creación del esquema del almacén

Las posibles modificaciones incluyen añadir o eliminar nuevas clases y atributos, además de identificar los elementos de las fuentes que deben ser integrados para obtener el valor de los diferentes atributos del almacén. En la parte derecha del área de trabajo hay dos listas de elementos que representan los atributos de la fuente seleccionada (a la izquierda) y los elementos de las fuentes de datos que se deben integrar para obtener un valor del atributo seleccionado del almacén (a la derecha). Para añadir un nuevo atributo de la

fuente seleccionada a la lista de elementos que se deben integrar basta con seleccionarlo y pulsar sobre el botón ">>".

Una vez determinado el esquema del almacén, también es posible modificar algunos de los parámetros temporales de la integración. En la Figura 80 se muestra el formulario que permite seleccionar el nivel de detalle temporal con el que se desean almacenar los valores de los diferentes atributos del esquema del almacén. Se muestra una lista de todas las fuentes de datos que intervienen a la hora de integrar el parámetro seleccionado en el esquema del almacén y se le da al diseñador del mismo la posibilidad de seleccionar el nivel de detalle dentro de unos los valores adecuados para ese dato.

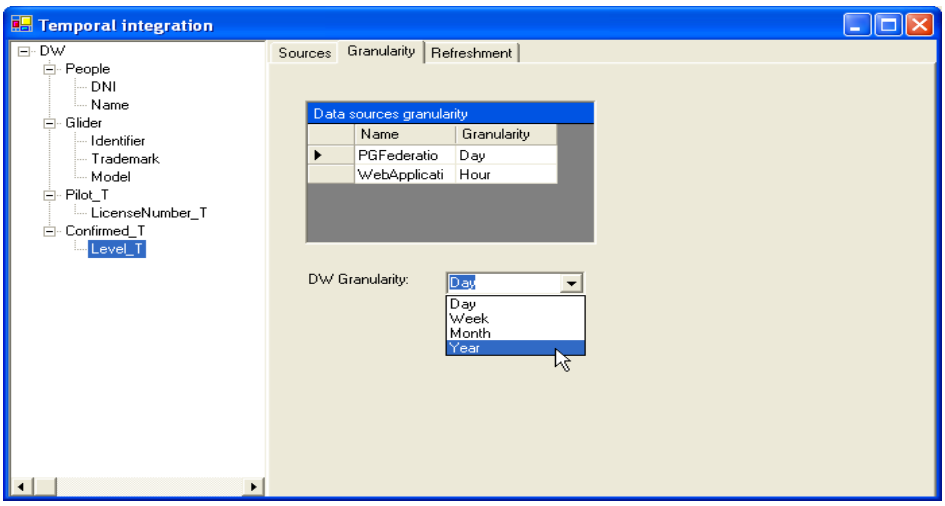

Figura 80. Elección de la granularidad

Por último, en la Figura 81 se muestra la forma en que el diseñador del almacén determina cuando se debe realizar el refresco de los datos. En la parte superior se muestra la ventana de disponibilidad de todas las fuentes de datos que intervienen en la integración del atributo del almacén seleccionado en el esquema. En el centro se muestra la intersección de todas las ventanas de disponibilidad de las fuentes implicadas en la integración. Justo debajo hay una serie de marcas que indican la posibilidad de realizar un refresco de los datos del almacén en ese preciso instante, determinadas automáticamente por el sistema. El diseñador del almacén puede entonces habilitar o deshabilitar cada uno de estos refrescos de datos mediante la lista de casillas de verificación que aparece en la parte inferior de la ventana.

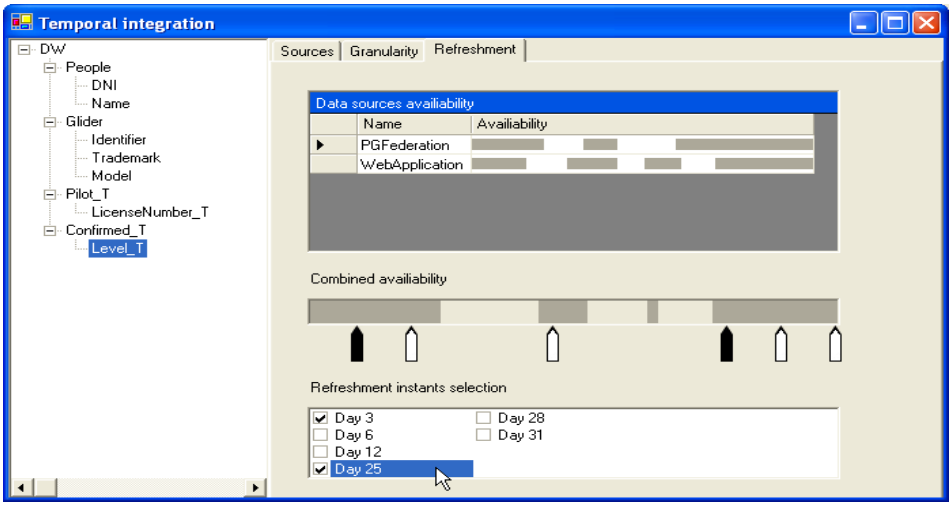

Figura 81. Refresco de los datos

BIBLIOGRAFÍA

[Adel95] Adelberg, B., García-Molina, H, Kao, B. Applying Update Streams in a Soft Real-Time Database System. Proceedings of the 1995 ACM SIGMOD, pages 245-256, 1995.

[Alag97] Suad Alagic. "The ODMG Object Model: Does it Make Sense?". En Proceedings of the 1997 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages & Applications OOPSLA 97. pp. 253-270. 1997

[Alag99] Suad Alagic. "Type-Checking OQL Queries In the ODMG Type Systems". En ACM Transactions on Database Systems. Vol. 24(3). pp. 319-360. 1990

[ANSI86] ANSI/X3/SPARC Database System Study Group: "Reference Model for DBMS Standardisation." ACM SIGMOD Record, Vol. 15, No. 1 (March 1986) 19-58.

[Araq02] Araque, F. (2002). Data Warehousing with regard to temporal characteristics of the data source. *IADIS WWW/Internet International Conference*. Lisboa, Portugal.13-15 November, 2002.

[Araq02a] Araque, F. (2002). Personalized data extraction with temporal constraints. *IADIS WWW/Internet International Conference*. Lisboa, Portugal.13-15 November, 2002.

[Araq03] Araque, F. & Samos, J. (2003). Data warehouse refreshment maintaining temporal consistency. *5th Intern. Conference on Enterprise Information Systems*, ICEIS´03.Angers. France.

[Araq03a] Araque, F. (2003a). Real-time Data Warehousing with Temporal Requirements. *Decision Systems Engineering, DSE'03* (in conjunction with the CAISE'03 conference). 16-20 June 2003, Klagenfurt/Velden, Austria.

[Araq03b] Araque, F. (2003b). Integrating heterogeneous data sources with temporal constraints using wrappers. *The 15th Conference On Advanced Information Systems Engineering*. Caise Forum. 16-20 June 2003, Klagenfurt, Austria.

[Araq06] Araque, F., Salguero, A., and Abad, M.M. 2006. Application of data warehouse and Decision Support System in Soaring site recommendation. *Proc. Information and Communication Technologies in Tourism*, ENTER 2006. Computer Springer Verlag, 18-20 January. Lausanne, Switzerland.

[Araq99] Araque, F & Samos, J. External Schemas in Real-Time Object-Oriented Databases. *The 20th IEEE Real-Time Systems Symposium. IEEE Computer Society*. WIP. ISBN 0-769-0475-2. 1-3 December, Phoenix, Arizona (USA).

[Aren96] Y. Arens, C. A. Knoblock, and W. Chen. Query reformulation for dynamic information integration. Journal of Intelligent Information System, 6:99-130, 1996.

[Ashi97] Ashish, N; Knoblock, C,: Wrapper Generation for Semi-structured Internet Sources, ACM SIGMOD Workshop on Management of Semi-structured Data, Tucson, Arizona. (1997).

[Berns00] A. Bernstein, A. Levy, R. Pottinger. A vision for management of complex models. Microsoft. TR 2000-53.

[Bert03] Bertino E., Ferrari E., Guerrini G., Merlo I., "T-ODMG: an ODMG compliant temporal object model supporting multiple granularity management", Information Systems, Volume 28 , Issue 8 (pg 885 - 927), December 2003.

[Bert92] Bertino, E. "A View Mechanism for Object-Oriented Databases". En Proceedings of 3rd International Conference on Extending Database Technology. EDBT 92. pp. 136-151. 1992

[Bert96] Bertino E., Ferrari E., Guerrini G., "A Formal Temporal Object-Oriented Data Model", Proceedings of the 5th International Conference on Extending Database Technology - EDBT'96, Avignon (France), 1996.

[Bert98] Bertino E., Ferrari E., Guerrini G., Merlo I., "Extending the ODMG Object Model with Time", Proccedings of the 12th European Conference on Object-Oriented Programming - ECOOP'98, Brussels (Belgium), July 20-24, 1998.

[Born99] M. Bornhövd, P. Buchmann. A prototype for meteadata based integration of internet sources. Proceedings of the Conference on Advanced Information Systems Engineering (CAISE'99), Heidelberg, Germany, 1999.

[Bruc02] Bruckner, Robert M. and Tjoa, A M. Capturing Delays and Valid Times in Data Warehouses - Towards Timely Consistent Analyses *Journal of Intelligent Information Systems* (JIIS), Vol. 19(2), pp. 169-190, Kluwer Academic Publishers, Sept, 2002.

[Calv98] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, R. Rosati. Source Integration in Data Warehousing. Fundations of Data Warehouse Quality (DWQ). Technical Report. DWQ-UNIROMA-002.

[Calv98] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, R. Rosati. Source Integration in Data Warehousing. Fundations of Data Warehouse Quality (DWQ). Technical Report. DWQ-UNIROMA-002.

[Cast93] M. Castellanos. Semiautomatic Semantic Enrichment for the Integrated Access in Interoperable Databases. PhD thesis, Departament Llenguatges i Sistemes Inform`atics, Universitat Polit`ecnica de Catalunya, Barcelona, June 1993.

[Cast94] Castellanos, M., Saltor, F., García, M.: A Canonical Model for the Interoperatibility among Object Oriented and Relational Models. In: Ozsu, T. et al. (eds.): Distributed Object Management (Edmonton, 1992), Morgan Kaufmann (1994), 309-314.

[Cate00] Catell, R.G.G. The Object Database Standard: ODMG 3.0. Morgan Kaufmann. 2000

[Cate94a] Catell, R.G.G. The Object Database Standard: ODMG-93. Morgan Kaufmann. 1994

[Cate94b] Catell, R.G.G. Object Data Management. Addison-Wesley. 1994

[Cate95] Catell, R.G.G. "Experience with the ODMG Standard". En StandardView Vol. 3(3). pp. 90-95. 1995

[Cate97] Catell, R.G.G. The Object Database Standard: ODMG 2.0. Morgan Kaufmann. 1997

[Chau97] Chaudhuri, S., Dayal, U. (1997). OLAP technology and data warehousing, ACM SIGMOD Records, Hewlett-Packard Labs, San Jose, 1997.

[Chaw94] S. Chawathe, H. García-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. In Proc. of IPSJ Conference, Tokyo (Japan), 1994.

[Chen00] TxnWrap: A transactional Approach to Data Warehouse Maintenance. CS TR Series. Worcester Polytechnic Institute. Dicembre, 2000.

[Clif00] Clifford, J., Dyreson, C., Isakowitz, T., Jensen, C. S., and Snodgrass, R. T. (1997). On the Semantics of "Now" in Databases. ACM Transactions on Database Systems, 22(2):171–214.

[Cres01] Valter Crescenzi, Giansalvatore Mecca, Paolo Merialdo. Road Runner: Towards Automatic Data Extraction from Large Web Sites. Technical Report RT-DIA-64-2001, Universita di Roma Tre..

[Datt96] Datta, A. Databases for Active Raidly Changing data Systems (ARCS): Augmenting Real-Time databases with Temporal and Active Characteristics. In Proceedings of the First International Workshop on Real-Time Databases, pp. 8-14, Marzo 1996.

[Datt96] Datta, A. Databases for Active Raidly Changing data Systems (ARCS): Augmenting Real-Time databases with Temporal and Active Characteristics. In Proceedings of the First International Workshop on Real-Time Databases, pp. 8-14, Marzo 1996.

[Dayal92] Dayal U., Wuu G.T.J., "A Uniform Approach to Processing Temporal Queries", Proceeding of International Conference on Very Large Database - VLDB'92, Vancouver (Canada), 1992.

[Devl97] Devlin, Barry. Data warehouse: from architecture to implementation Addison Wesley, 1997.

[Doan00] Learning Source Descriptions for Data Integration. Proceedings of the Third International Workshop on the Web and Databases (WebDB-2000), pages 81-86, 2000. Dallas, TX: ACM SIGMOD.

[Duma04] M. Dumas, M.C. Fauvet, P.C. Scholl. TEMPOS: A Platform for Developing Temporal Applications on top of Object DBMS. IEEE Transactions on Knowledge and Data Engineering 16(3):354-374, March 2004. IEEE Computer Society.

[Elma97] Elmasri, R., Navathe, S.B. Sistemas de Bases de Datos. Conceptos Fundamentales. Addison-Wesley. 1997

[Engs00] Engström, H., Chakravarthy, S. and Lings, B. (2000) A Holistic Approach to the Evalaution of Data Warehouse Maintenance Policies, Technical Report, HS-IDA-00-001, University of Skövde, January.

[Garc95] M. García-Solaco, F. Saltor, M. Castellanos. A semantic-discriminated approach to integration in federated databases. Proc. Of the 3rd Int. Conf. On Cooperative Information Systems (CoopIS-95), pages 19-31, 1995.

[Garc98] H. Garcia-Molina, W. J. Labio, and J. Yang. Expiring Data in a Warehouse." In Proceedings of the 24th VLDB Conference, New York, August, 1998.

[Gatz99] S. Gatziu, A. Vavouras. Data Warehousing: Concepts and Mecahnisms. Informatik (Zeitschrift der Schweizerischen Informatikorganisationen) 0:1, February 1999.

[Goer93] Göers, J., Heuer, A. "Definition and Application of Metaclasses in an Object-Oriented Database Model". En Proceedings of the Ninth International Conference on Data Engineering. ICDE 93. pp. 373-380. 1993

[Grif04] Tony Griffiths, Alvaro A. A. Fernandes, Norman W. Paton, Robert Barr: The Tripod spatio-historical data model. Data Knowl. Eng. 49(1): 23-65 (2004).

[Grus98] Gruser J.-R., Raschid L., Vidal M.E., Bright L. Wrapper Generation for Web Accessible Data Sources, Proceedings of Third IFCIS International Conference on Cooperative Information Systems (CoopIS), 1998.

[Haas98] Haas L., Kossman D., Wimmers E., Yang J.: "Optimizing Queries across Diverse Data Sources", 23rd Very Large Data Bases, August 1998, Athens, Greece.

[Hall00] M. Haller, B. Pröll, W. Retschitzegger, A M. Tjoa, R.R. Wagner, Integrating Heterogeneous Tourism Information in TIScover – The MIRO-Web Approach. ENTER 2000, Information and Communication Technologies in Tourism, Barcelona, April 26-28, 2000.

[Hamm95] J. Hammer, H. García-Molina, J. Widom, W. Labio, Y. Zhuge. The stanford data warehousing project, IEEE Data Engineering 18(2), 41-48, 1995.

[Huhn93] M. N. Huhns, N. Jacobs, T. Ksiezyk, W. Shen an M. P. Singh, and P. E. Cannata. Integrating enterprise information models in Carnot. In Proc. of the Int. Conf. on Cooperative Information System (CoopIS-93), pages 32-42, 1993.

[Inmo02] Inmon, W: "Building the Data Warehouse", John Wiley&Sons, Third edition, 2002.

[Inmo92] Inmon, W.: "What is a Data Warehouse?." PRISM Tech Topic, Vol.1, No.1 (1992)

[Inmo93] Inmon,W.: "The Operational Data Store." PRISM Tech Topic, Vol.1, No.17 (1993)

[Inmo98] W. Inmon, C. Imhoff, R. Sousa. Corporate Information Factory. John Wiley&Sons, Inc. (1998).

[Jark00] Jarke, M., Lenzerini, M., Vassiliou, Y., and Vassiliadis, P., editors (2000). Fundamentals of Data Warehousing. Springer-Verlag.

[Jens92] Jensen, Clifford, Gadia, Segev, Snodgrass, 1992. A Glossary of Temporal Databese Concepts. SIGMOD Record (1992).

[Kako96] Kakoudakis I., Theodoulidis B., "The TAU Temporal Object Model", Technical Report TR-96-4, TimeLab, University of Manchester (UMIST), 1996.

[Kao95] Kao, B., Garcia-Molina, H.: "An Overview of Real-Time Database Systems." In S. Son (Ed.), Advances in Real-Time Systems, chapter 19. Prentice Hall, 1995.

[Kim94] Kim, W. "Observations on the ODMG-93 Proposal for an Object-Oriented Database Language". En SIGMOD Record. Vol. 23(1). pp. 4-9. 1994

[Kurz98] Kurz, A.; Tjoa, M, (1998), Integrating Executive Information Systems and Data Warehouse, Proceedings 2nd Int. Conference on Business Information Systems - BIS98, April 22-24, 1998, Poznan, Poland.

[Labi97] W. J. Labio, Y. Zhuge, J. L. Wiener, H. Gupta, H. García-Molina, J. Widom. The WHIPS Prototype for Data Warehouse Creation and Maintenance. Proceedings of SIGMOD, pages 557--559, 1997.

[Levy95] Alon Y. Levy, Divesh Srivastava and Thomas Kirk. Data model and query evaluation in global information systems. Journal of Intelligent Information Systems (JIIS), 5(2): 121-143, 1995. Special Issue on Networked Information Discovery and Retrieval.

[Levy95] Alon Y. Levy, Divesh Srivastava and Thomas Kirk. Data model and query evaluation in global information systems. Journal of Intelligent Information Systems (JIIS), 5(2): 121-143, 1995. Special Issue on Networked Information Discovery and Retrieval.

[Levy96] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. Proc. VLDB 1996, pp. 251--262.

[Levy99] Levy, A.Y.: Combining Artificial Intelligence and Databases for Data Integration, special issue of LNAI: Artificial Intelligence Today; RecentTrends and Developments, 1999.

[Levy99] Levy, A.Y.: Combining Artificial Intelligence and Databases for Data Integration, special issue of LNAI: Artificial Intelligence Today; RecentTrends and Developments, 1999.

[Lin94] Lin, K.; Son, S.H.: "Real-Time Databases: Characteristics and Issues". IEEE Workshop for Object-Oriented Real-Time Dependable Systems, Irvine, CA, October 1994.

[Mich97] G. De Michelis, E. Dubois, M. Jarke, F. Matthes, J. Mylopoulos, M. Papazoglou, K. Pohl, J. Schmidt, C. Woo, E. Yu. "Cooperative Information Systems: A Manifesto." *In M. Papazoglou, G. Schlageter (Eds.) Cooperative Information System: Trends and Directions, Academic Press*, 1997.

[Mour04] Moura Pires, J., Pantoquilho, M., & Viana, N. (2004). Real-Time Decision Support System for Space Missions Control. Int. Conference on Information and Knowledge Engineering, Las Vegas, USA.

[Oliv01] Oliva, M., and Saltor, F. Integrating Multilevel Security Policies in Multilevel Federated Database Systems. In B. Thuraisingham, R. van de Riet, K.R. Dittrich, and Z. Tari, editors, Data and Applications Security: Developments and Directions, pages 135–147. Kluwer Academic Publishers, Boston, 2001.

[Oliv96] M. Oliva, F. Saltor: "A Negotiation Process Approach for Building Federated Databases". In: 10th ERCIM Database Research Group Workshop on Heterogeneous Information Management, Prague 1996, ERCIM-96-W003 CRCIM, 1996, pp 43-49.

[Rama93] K. Ramamritham. "Time for Real-Time Temporal Databases?" In Proc. Int'l. Workshop on an Infrastructure for Temporal Databases (June, 1993).

[Rama93] K. Ramamritham. "Time for Real-Time Temporal Databases?" In Proc. Int'l. Workshop on an Infrastructure for Temporal Databases (June, 1993).

[Rodr97] E. Rodriguez, M.Oliva, F. Saltor, B. Campderrich: "On Schema and Functional Architectures for Multilevel Secure and Multiuser Model Federated DB Systems". In S. Conrad et al. (eds), Int'l CAiSE'97 Workshop on Engineering Federated Database Systems (EFDBS'97), pp 93-104. Otto-von-Guericke-Universitat Magdeburg, Fakultat fur Informatik, 1997.

[Rund00] Elke A. Rundensteiner, Andreas Koeller, and Xin Zhang. Maintaining Data Warehouses over Changing Information Sources. Communications of the ACM. June 2000. Special Section on System Integration.

[Rund92a] Rundensteiner, E. A. "Multiview: A Methodology for Supporting Multiple Views in Object-Oriented Databases". En Proceedings of the 18th International Conference on Very Large Databases. VLDB 92. pp. 187-198. 1992

[Samo97] Samos, J., Saltor, F.: "External Schemas in a Schema-Evolution Environment for OODBs." In R. Wagner (Ed.), Proc. of the Int'l Workshop on Database and Expert Systems Applications (Toulouse, Sep. 1997), IEEE Computer Society Press, pp. 516-522.

[Samo98] Samos, J., Saltor, F., Sistac, J., Bardés, A.: "Database Architecture for Data Warehousing: An Evolutionary Approach." In G. Quirchmayr et al. (Eds.): Proc. Int'l Conf. on Database and Expert Systems Applications (Vienna, Aug. 1998), Springer-Verlag, pp. 746-756.

[Samo98a] Samos, J., Saltor, F.: "Integration of Derived Classes in Object Schemas." IADT'98 – Int'l. Workshop on Issues and Applications of Database Technology (Berlin, July 1998)

[Samo99] Samos, J. ; Abelló, A.; Oliva, M.; Rodríguez, E.; Saltor, F.; Sistac, J.; Araque, F.; Delgado, C.; Garví, E. ; Ruiz, E.: Sistema Cooperativo para la Integración de Fuentes Heterogéneas de Información y Almacenes de Datos. Novática, #142 (Nov-Dec 1999), pp. 44-49.

[Scho91a] Scholl, M.H., Schek, H.J "Supporting Views in Object Oriented Databases". En IEEE Database Engineering Bulletin. Vol 14(2) pp. 43-47. 1991

[Shet90] Sheth, A., Larson, J.: "Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases." ACM Computing Surveys, Vol. 22, No. 3 (Sep. 1990)

[Snod86] Snodgrass, R., Ahn, I.: "Temporal Databases". IEEE Computer, Vol. 19, Nº 9, September 1986, pp.35-42.

[Stan99] Stankovic, J., Son, S., Hansson, J.. "Misconceptions About Real-Time Databases". IEEE Computer, Vol. 32, Nº 6, June 1999, pp. 29-36.

[Stan99] Stankovic, J., Son, S., Hansson, J.. "Misconceptions About Real-Time Databases". IEEE Computer, Vol. 32, Nº 6, June 1999, pp. 29-36.

[Su91] Su S., Chen H., "A Temporal Knowledge Representation Model OSAM*/T and its query language OQL/T", Proceedings of the 17th International Conference on Very Large Databases - VLDB'91, Barcelona (Spain), September 1991.

[Torr00] Torres, M. Samos, J. "Definition of External Schemas in ODMG Databases" En Proceedings of the 6th International Conference on Object Oriented Information Systems. OOIS 2000. pp. 3-14. 2000

[Torr01a] Torres, M. Samos, J. "Generation of External Schemas in ODMG Databases" En Proceedings of the 5th International Database Engineering and Applications Symposium 2001. IDEAS 2001. pp 89-98. 2001

[Torr01b] Torres, M. Samos, J. "Closed Schemas in Object-Oriented Databases". En Proceedings of the 12th International Conference of Database and Expert Systems Applications 2001. DEXA 2001. pp. 826-835. 2001

[Torr01c] Torres, M. Samos, J. "Una propuesta de extensión de los metadatos de ODMG para la definición de esquemas externos". En Actas de las VI Jornadas de Ingeniería del Software y Bases de Datos. JISBD 2001. pp. 507-521. 2001

[Torr02] Torres, M. Samos, J. "Extending ODMG Metadata to Define External Schemas". Technical Report LSI-2002-1. Departamento de Lenguajes y Sistemas Informáticos. Universidad de Granada. 2002

[Torr02a] Manuel Torres. Definición de esquemas externos en bases de datos ODMG. Tesis Doctoral. Dpto. Lenguajes y Computación, Universidad de Almería. Mayo 2002.

[Tres92] Tresch, M, Scholl, M.H. "Meta Object Management and its Application to Database Evolution" En Proceedings of the 11th International Conference on the Entity-Relationship Approach. ER 92. pp. 299-321. 1992.

[Tres93] M. Tresch, M. Scholl, "Schema Transformation without Database Reorganization," *SIGMOD Record*, vol. 22, no. 1, pp. 21-27, 1993.

[Vavo99] A. Vavouras, S. Gatziu, K.R. Dittrich. Modeling and Executing the Data Warehouse Refreshment Process. Proc. of the 1999 Intl. Symposium on Database Applications in Non-Traditional Environments (DANTE '99), Kyoto, Japan, November 1999.

[Wido95] Widom, J, Research Problems in Data Warehousing, Proceedings of 4th Intl. Conference on Information and Knowledge Management (CIKM), 1995.

[Wido95] Widom, J, Research Problems in Data Warehousing, Proceedings of 4th Intl. Conference on Information and Knowledge Management (CIKM), 1995.

[Wuu93] Wuu G., Dayal U., "A Uniform Model for Temporal and Versioned Object-oriented Databases", Temporal Databases: Theory, Design, and Implementation, Benjamin/Cummings, pp 230-247, 1993.

[Zach00] Zachary, G., Levy, A., Weld, D., Florescu, D., Friedman, M., (2000), Adaptive Query Processing for Internet Applications, IEEE Data Engineering Bulletin, June.

[Zhou96] G. Zhou, R. Hull, and R. King. Generating data integration mediators that use materializations. Journal of Intelligent Information Systems, 6:199-221, 1996.

[Zhug96] Y. Zhuge, H. Garcia-Molina, J. Wiener. The Strobe Algorithms for Multi-Source Warehouse Consistency. In Proceedings of Fourth International Conference on Parallel and Distributed Information Systems, 1996.