MEDIADOR PARA LA RECUPERACIÓN E INTEGRACIÓN DE DATOS MÉDICOS DE FUENTES HETEROGÉNEAS

GUILLERMO ANDRES MONTOYA GARCIA
0859167
Autor
guillermo.montoya@correounivalle.edu.co

Trabajo para optar el título de Ingeniero de Sistemas

MARTHA MILLAN
Directora
martha.millan@correounivalle.edu.co

MARIA CONSTANZA PABON
Codirectora
mcpabon@javerianacali.edu.co

Facultad de Ingeniería
Escuela de Ingeniería de Sistemas y Computación
Programa Académico de Ingeniería de Sistemas
Proyecto de Ingeniería de Sistemas
Tuluá, Junio 21 de 2013

Nota de aceptación:

_____
_____
_____
_____
_____
_____

_____
Firma del presidente del jurado

_____
Firma del jurado

_____
Firma del jurado

Tuluá, 21 de Junio del 2013

Agradecimientos

Me gustaría dar un agradecimiento especial a todas esas personas que hicieron posible esto.

En primer lugar quisiera dar gracias a mi familia, por su apoyo incondicional durante este tiempo. A mis padres por darme el soporte para estar en una Universidad a mis hermanas por ser parte de mi vida. A ellos gracias por comprenderme en muchos momentos y por darme ese impulso para seguir adelante.

A los profesores que impartieron su conocimiento sobre mí. Gracias por su dedicación y por darme la oportunidad de formarme como profesional.

Gracias a la profesora María Constanza Pabón por darme la oportunidad de realizar este proyecto. Gracias por depositar su confianza en mí, por entregarme parte de su trabajo y por creer en que podría sacarlo adelante. A mi directora Martha Elena Millán gracias por su apoyo, paciencia y compresión durante la realización de este trabajo. A las dos gracias sobre todo por haber compartido su experiencia y conocimiento durante todo este tiempo. Mil y mil gracias.

A mis amigos y compañeros que hicieron parte de esta expedición de vida, gracias por todos los momentos vividos.

Y por último gracias a Dios por darme la oportunidad de culminar este trabajo. Y a todos los que de alguna forma tuvieron que ver con esto.

Índice General

Índice de figuras

Índice de tablas

Índice de anexos

Capítulo 1

1.1 Introducción

La información es uno de los motores más potentes que existen en la sociedad actual y una variable que influye, por ejemplo, en nuestras decisiones, en la vida de un paciente o en el funcionamiento de una empresa. Por eso, es fundamental que esa información (siempre en continuo movimiento) tenga una calidad óptima cuando llega al destinatario "necesitado de la información". Pero encontrar información para una necesidad específica es difícil. Esto se debe, en parte, al creciente volumen de la información disponible. Un segundo problema es su fragmentación y la proliferación de fuentes de información.

En el campo médico, generalmente es necesario integrar información de diversas fuentes ya que los datos de un paciente pueden estar almacenados en varios formatos (e.g. imágenes médicas, reportes de diagnóstico y datos de la historia clínica). La integración de esta información es importante para apoyar procesos de diagnóstico, enseñanza e investigación.

El trabajo que aquí se presenta forma parte de un proyecto de tesis doctoral cuyo propósito es desarrollar una arquitectura de integración de datos que sirva de base para implementar un lenguaje conceptual de recuperación de datos médicos. Como parte de la arquitectura de integración, se requiere implementar un mediador y dos *wrappers*. El mediador es el componente encargado de unificar el acceso a las diferentes fuentes, brindando una interfaz por medio de la cual se gestionan las consultas del usuario. El mediador descompone la consulta en subconsultas que puede responder la fuente de datos correspondiente. Los *wrappers* se encargan de traducir la subconsulta del lenguaje del mediador al lenguaje propio de cada una de las fuentes. Para esta arquitectura, inicialmente, se desarrollarán dos *wrappers*, uno para acceder a datos en una base de datos relacional y el otro para acceder a datos e imágenes almacenados bajo el estándar DICOM. A partir de las subconsultas dirigidas a las fuentes se obtienen diferentes resultados que el mediador debe integrar para entregarla al usuario.

En este documento se presenta el desarrollo de un prototipo de mediador de la arquitectura de integración de datos mencionada.

1.2 Planteamiento y Formulación del Problema

Actualmente en el campo médico la información digital se ha incrementado de manera significativa. Las entidades prestadoras de salud recopilan datos en sus sistemas de información, como resultado de las labores diarias de atención a los pacientes. Estos datos pueden ser un recurso valioso para propósitos de enseñanza, planeación, control de tratamientos, diagnóstico e investigación. Sin embargo, obtener información relevante para satisfacer una necesidad de un usuario médico implica acceder a diferentes tipos de fuentes y formatos (imágenes médicas, datos clínicos de pacientes, datos demográficos). Generalmente, las fuentes tienen estructuras diferentes y esto dificulta recuperar información que relacione los distintos datos que se requieren. Se hace entonces necesario integrar los datos. Generalmente, cuando un médico atiende a un paciente

necesita tener acceso a la información completa, que puede provenir de distintas fuentes. Por lo que se hace necesario proveer herramientas que faciliten acceder a estos datos.

Como respuesta a este problema se está desarrollando una tesis doctoral "Lenguaje para la Recuperación de Datos Heterogéneos en el Dominio Médico [28], con el propósito de permitir a los usuarios formular consultas de manera fácil, con una sintaxis simple y con la expresividad suficiente para expresar consultas del dominio médico. Para el desarrollo de este lenguaje, se utiliza una arquitectura de integración basada en mediación, que se muestra en la figura 1 [1]. Hacen parte de la arquitectura una interfaz de usuario, un mediador, *wrappers* y anotaciones asociadas a las diversas fuentes de datos.

Este trabajo de grado se enmarca en esta propuesta doctoral y, particularmente, tiene como objetivo construir un prototipo funcional del mediador, que ofrezca una interfaz única para acceder a las diferentes fuentes de datos y que permita recuperar e integrar información relevante de diferentes fuentes a partir de una consulta médica. Este componente cuenta con un modelo de datos conceptual (modelo global de la arquitectura) que representa la información disponible en las diferentes fuentes, un vocabulario que ayuda a expandir los términos de una consulta con base en una ontología de dominio y unas reglas de mapeo que representan la forma de obtener las entidades del modelo global en las en fuentes de datos.

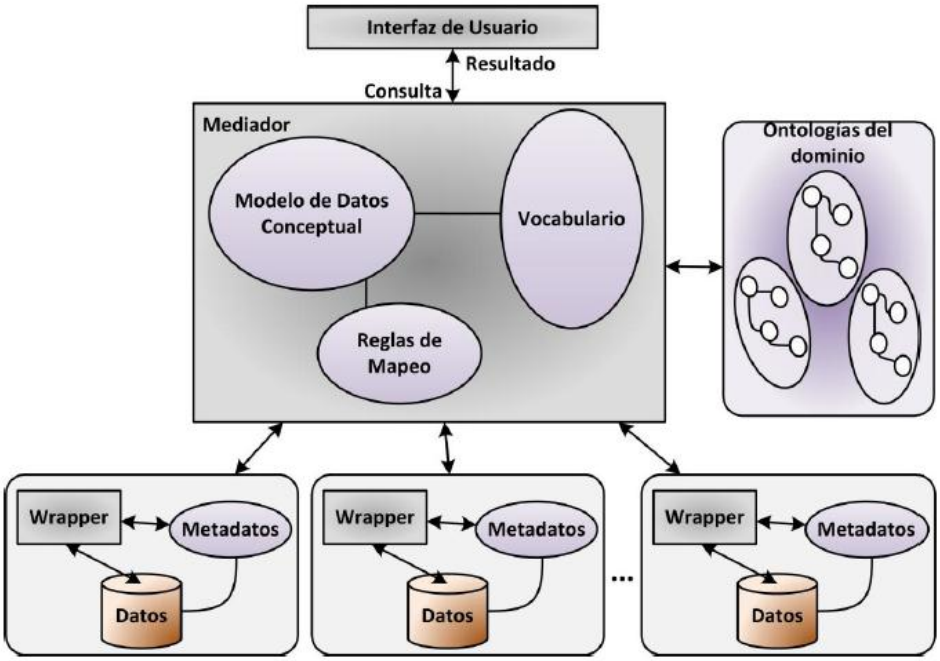


Figura 1: Arquitectura basada en mediación

El desarrollo del trabajo debe permitir dar respuesta a la siguiente pregunta:

¿Cómo identificar las fuentes a las que se debe acceder para construir la respuesta a una consulta expresada en un lenguaje lógico?

Sistematización del problema

-¿Cómo identificar las fuentes de datos donde se encuentra información relevante para una consulta médica?
-¿Cómo descomponer la consulta en subconsultas a las fuentes que contienen información relevante?
-¿Cómo representar los mapeos entre el modelo global y los modelos locales de las fuentes de datos DICOM y relacional?
-¿Cómo representar la consulta en una estructura en memoria?

1.3 Objetivos

Objetivo General

Construir un prototipo de mediador para acceder a datos de fuentes heterogéneas relevantes a una consulta en el campo médico, por medio de un modelo conceptual de datos, apoyado en un vocabulario y reglas de mapeo entre el modelo global y los esquemas de las fuentes de datos

Objetivos Específicos

1. Representar los mapeos entre el modelo global y los modelos locales de las fuentes de datos (DICOM y relacional) mediante un enfoque GAV (Global-as-View)
2. Evaluar algoritmos de identificación de las fuentes involucradas en una consulta y de subdivisión de consultas a partir de una especificación definida de mapeos.
3. Implementar los algoritmos de identificación de fuentes y de subdivisión de consultas seleccionados.

1.4 Organización del documento

El resto del documento está organizado en cuatro capítulos. En el segundo capítulo se presenta un marco teórico general que incluye los conceptos teóricos básicos relacionados con el modelo de integración propuesto. Inicialmente, se describen las arquitecturas comúnmente usadas para resolver el problema de integración de datos y sus componentes, haciendo énfasis en las arquitecturas basadas en mediación. Una breve descripción de las bases de datos orientada a grafos se presenta por considerarse en este trabajo como representación del modelo global. Finalmente, se hace revisión de algunos motores o librerías para el manejo de estas bases de datos y se describe también, de forma general, algunas de las herramientas que dan soporte al desarrollo de este trabajo. Algunos prototipos de arquitecturas basadas en mediación que se han propuesto, se describen también. Por último, se definen algunos términos utilizados en el documento.

El tercer capítulo, presenta el modelo que se plantea como solución al problema. En cuarto capítulo se describe el funcionamiento del mediador y se presentan los resultados de las pruebas de funcionamiento. Finalmente, en el capítulo quinto se discuten los resultados del trabajo y se proponen algunas líneas de trabajo futuro.

Capítulo 2

Este capítulo presenta un marco teórico general relacionado con las áreas con las cuales tiene que ver este estudio y una revisión de los trabajos propuestos en la literatura especializada que se relaciona con el problema objeto de estudio de este trabajo de grado. El capítulo está organizado en tres secciones. En la primera sección, se presenta el marco teórico. En la segunda y tercera sección, respectivamente, se revisan los antecedentes del trabajo y se describe el marco conceptual.

2.1 Marco teórico

2.1.1 Integración de datos de fuentes heterogéneas

La integración de datos busca proveer a los usuarios de una vista unificada, integrada y global de datos que se encuentran en fuentes diversas, heterogéneas y autónomas. De esta manera, los usuarios pueden formular consultas que el sistema ejecuta de manera transparente. La vista puede ser materializada o virtual, materializada cuando se replican los datos de las diversas fuentes en un repositorio central y virtual cuando los datos se acceden en sus fuentes locales.

2.1.2 Enfoques utilizados para implementar la integración

Desde el punto de vista de cómo se implementa la integración existen los siguientes enfoques*: Data translation* (integración materializada), *Query translation* (integración virtual) e *Information Linkage* [1].

En el enfoque ***Data translation,*** la vista se materializa y se replican los datos de las diferentes fuentes en un repositorio central adecuándolos a un esquema unificado. Bajo este enfoque la actualización de una fuente puede conducir a inconsistencias porque un cambio en una fuente no se refleja inmediatamente en el repositorio central, es decir se podría consultar información desactualizada. También se requiere mayor capacidad de almacenamiento porque en el almacén central se duplica la información de las fuentes. Debido a que toda la información está disponible en un punto central, el sistema puede reaccionar mucho mejor a consultas arbitrarias del usuario a partir del conjunto integrado de información.

La figura 2, tomada de [2], muestra una arquitectura materializada en la cual un módulo de extracción de datos se encarga de replicar los datos de las diferentes fuentes en el almacén de datos (*Data WareHouse*). El usuario accede a los datos usando una aplicación que se conecta al almacén de datos (*Data WareHouse*).

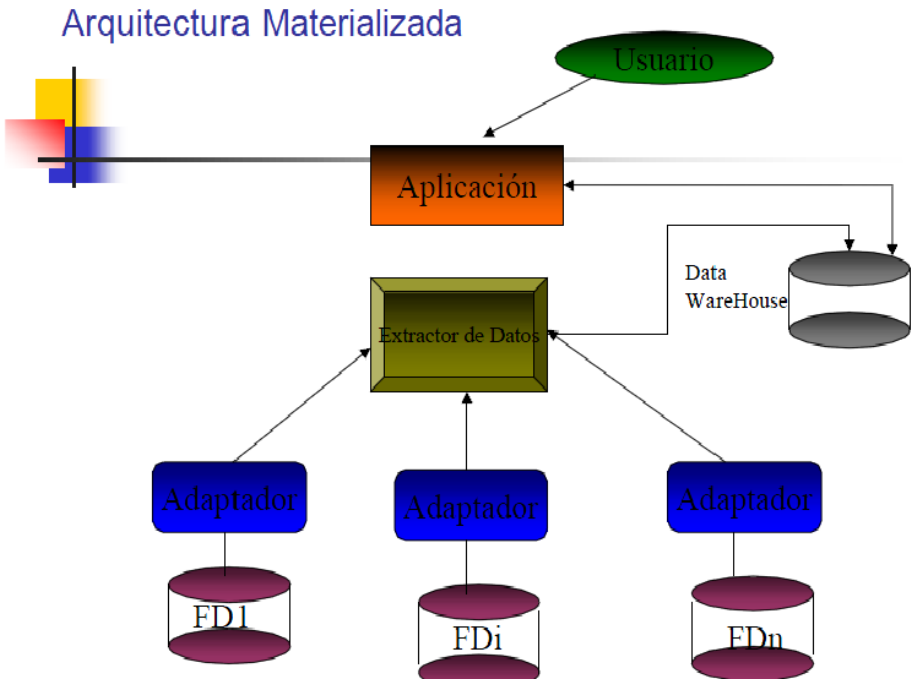


Figura 2: Enfoque *Data Translation*- Arquitectura Materializada

***Information Linkage*:** Las aplicaciones de software típicamente hoy en día están escritas en lenguaje orientado a objetos y acceden a datos almacenados en bases de datos relacionales. El mapeo de estas aplicaciones a las bases de datos requiere integrar el esquema de la aplicación y el esquema de la base de datos. Esto puede resultar una tarea complicada dependiendo de qué tan diferentes sean los esquemas en su construcción. Una forma de solucionar este problema es usar el enfoque *Information Linkage*, donde se crean referencias entre los datos de las diferentes fuentes que corresponde a un mismo objeto del mundo real [1].

El enfoque ***Query translation,*** usa una vista virtual y provee mecanismos que permiten acceder a los datos en sus respectivos repositorios. Los datos se encuentran lógicamente integrados y por lo tanto, no existe el riesgo de que las actualizaciones que se realicen en una fuente afecten la integridad del sistema. Sin embargo, el acceso integrado de los datos es costoso computacionalmente. Este enfoque a menudo se denomina mediador de consultas o EII (*Enterprise Information Integration*). Una consulta se traduce en subconsultas que se ejecutan sobre cada una de las fuentes de datos. Los resultados de cada una de las fuentes se integra para responder la consulta original, de modo que parece haber venido de una base de datos integrada.

En este enfoque se utiliza comúnmente una arquitectura basada en mediación que incluye una interfaz de usuario, un mediador, *wrappers* y las fuentes de datos. La interfaz de usuario captura la consulta y se la entrega al mediador. El mediador procesa globalmente la consulta con base en un modelo de datos global y en los mapeo entre este y los esquemas de las fuentes locales y genera subconsultas que se envían a las fuentes de datos pertinentes. El lenguaje en que se generan las subconsultas depende del mediador y puede llegar a ser diferente de los lenguajes de consulta de las diferentes fuentes

locales. Si las subconsultas se generan en un lenguaje diferente, un *wrapper* se encargará de hacer la traducción para que la consulta sea entendida en las diferentes fuentes locales. Finalmente, después de que las consultas se ejecutan en cada fuente local, el mediador recibe los resultados, los integra y se los entrega al usuario.

En la figura 3, tomada de [1], se muestra una arquitectura genérica basada en mediación, que cuenta con un mediador, *wrappers* para cada fuente de datos y una interfaz de usuario.

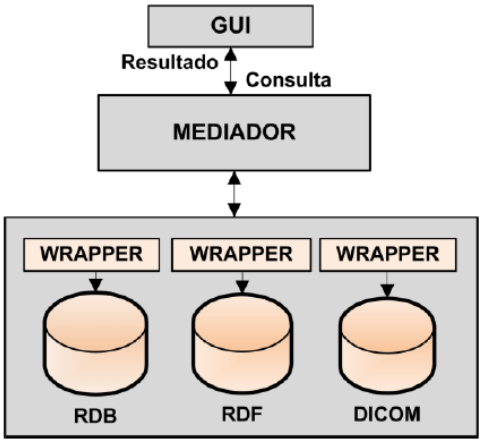


Figura 3:Enfoque Query Translation- Arquitectura basada en mediación

2.1.3 Arquitectura Basada en mediación

Como se mencionó anteriormente la arquitectura basada en mediación generalmente consta de 4 elementos: la interfaz de usuario, el mediador, *wrappers* y fuentes de datos. A continuación se definirán con más detalle sus componentes.

Interfaz de Usuario

La interfaz recibe las consultas que vienen del usuario. El usuario puede describir estas consultas, por ejemplo, mediante lenguajes de consulta de nivel lógico como SQL. Este tipo de lenguajes permite definir consultas complejas, sin embargo la formulación de consultas es difícil para usuarios no técnicos puesto que requiere conocer la estructura de almacenamiento de los datos. Las consultas se pueden expresar, además, mediante lenguajes de consulta de nivel conceptual que le permiten al usuario expresar sus consultas con un mayor nivel de abstracción, de forma cercana como lo hace en su dominio y usando términos específicos del mismo. Sin embargo, existen muy pocos lenguajes que trabajen a este nivel. Otras interfaces de usuario que también se pueden encontrar son los buscadores para acceder a recursos que están en la web y las interfaces que reciben imágenes de ejemplo como criterio de búsqueda.

Mediador

El mediador es un componente de software que muestra una única interfaz para las diversas fuentes de datos. El mediador se encarga del procesamiento global de la consulta que se realiza con base en el modelo de datos global y los mapeos entre este y los esquemas de datos locales de las fuentes que integra. En el procesamiento global, el mediador descompone las consultas en subconsultas de acuerdo con las fuentes de datos

disponibles y sus capacidades. Estas subconsultas se procesan en las respectivas fuentes de datos, el mediador recibe los resultados, los integra y los entrega al usuario.

Para realizar el mapeado entre los esquemas de datos locales y el modelo global existen dos aproximaciones**: punto de vista global (*Global-As-View-GAV*)** o **punto de vista local (*Local-As-View-LAV*)** [1]. En el primero, el modelo global se define a partir de los esquemas locales, cada uno de sus elementos se mapea a una vista que combina datos de los esquemas locales. En el segundo, el modelo global es independiente de los esquemas locales, cada elemento de los esquemas locales se mapea a una vista del modelo global. Este punto de vista permite la extensibilidad de los sistemas ya que se pueden agregar nuevas fuentes locales sin afectar el modelo global, pero al dividir la consulta en subconsultas se hace más complejo encontrar las fuentes locales que pueden aportar respuestas. Algunos autores han propuesto enfoques híbridos, como ***GLAV*** [3] que toma las ventajas que ofrece tanto ***LAV*** como ***GAV***.

Wrapper

El *wrapper* es el componente de software que encapsula el acceso a una fuente de datos. Toma la consulta en el lenguaje propio del mediador y la convierte en una consulta que la fuente es capaz de procesar. Para ello usa los metadatos, el modelo de datos lógico de la fuente y un conjunto de reglas de traducción que permiten reescribir la consulta en términos del lenguaje que el esquema local reconoce. Los metadatos son datos sobre los datos que permiten ampliar la semántica. El modelo lógico contiene una especificación de los datos almacenados el más usado en las fuentes de datos es el relacional.

2.1.4 Bases de datos orientadas a grafos

Una base de datos orientada a grafos usa una estructura de un grafo con nodos, aristas y propiedades para representar y almacenar datos. Una definición más técnica de una base de datos orientada a grafos es cualquier sistema de almacenamiento que provee índice libre de adyacencia. Esto significa que cada elemento contiene un puntero directo a su elemento adyacente y no son necesarias operaciones de búsqueda de índices.

Estructura

Una base de datos orientada a grafos está basada en la teoría de grafos, emplea nodos y arcos. Los nodos son similares en naturaleza a los objetos en la programación orientada a objetos. En algunos modelos, los nodos (y/o los arcos) tienen la capacidad de almacenar atributos.

En la figura 4, tomada de [12], se muestra la estructura básica de una base de datos orientada a grafos. Los nodos representan entidades tales como personas, empresas, cuentas o cualquier otra entidad que se quiera representar. Las propiedades son información pertinente que se refiere a los nodos. Por ejemplo, si se tuviera “Wikipedia” como un nodo, éste se podría ligar a una propiedad “sitio web”, “material de referencia” o “palabra que empieza con la letra w”, esto depende de cuáles aspectos se estén modelando en la base de datos. Las aristas son líneas que conectan nodos con otros nodos, o nodos con propiedades y representan relaciones entre las dos partes conectadas.

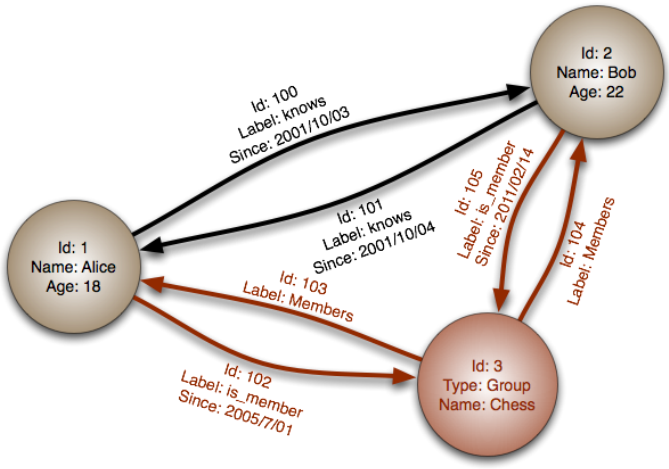


Figura 4: Estructura de una base de datos orientada a grafos

Las siguientes son algunas propiedades de las bases de datos orientadas a grafos [12]:

- Comparadas con las bases de datos relacionales, las bases de datos orientadas a grafos son a menudo más rápidas para gestionar conjuntos de datos asociativos además de que mapean directamente a la estructura de aplicaciones orientadas a objetos.
- Si el objetivo principal es analizar las relaciones, este tipo de bases de datos es el ideal ya que en el modelo relacional se tendrían que realizar demasiadas operaciones de join.
- Las bases de datos orientadas a grafos proveen herramientas poderosas para la ejecución de operaciones comunes sobre grafos. Por ejemplo, calcular el camino más corto entre dos nodos en un grafo, hacer búsqueda en profundidad o en anchura, entre otros.

2.1.5 Proyectos de bases de datos orientadas a grafos

Neo4j [19]

Es una base de datos orientada a grafos que almacena atributos en los nodos y en los arcos. Es una herramienta de código abierto, implementada en java.

Principales características

- Tiene un modelo orientado a grafos para la representación de datos.
- Los datos son almacenados en nodos conectados por relaciones. Las relaciones son tipadas y dirigidas
- Es un gestor de almacenamiento nativo, optimizado para el almacenamiento de estructuras de grafos
- Se puede desplegar como un servidor completo o una base de datos muy pequeña

- Cuenta con una API orientada a objetos
- Ofrece un *framework* transversal para consultas sobre grafos
- Soporta transacciones ACID (Atomicidad, Consistencia, Insolación, Durabilidad)
- Provee indexación para cada nodo o relación. Cada nodo o relación del grafo pueden tener un índice para facilitar su búsqueda.
- Provee un API REST que facilita la integración con tecnologías web como XML y Jason Provee enlaces con otros lenguajes (Pyhton, ruby, clojure, Scala, entre otros).
- Ofrece un paquete para hacer consultas en SPARQL
- Ofrece un paquete de implementaciones de algunos algoritmos comunes de grafos (búsqueda de caminos, camino más corto entre dos nodos, entre otros.)

InfiniteGraph

Es una base de datos de grafos distribuida implementada en java. No es una herramienta de código abierto y está disponible en versiones libres y pagas [20].

Principales características [21]

- Ofrece un API de bases de datos distribuidos de grafos
- Cuenta con visualizador para exportar a GraphML y JSON
- Implementa procesamiento multihilo
- Su modelo es un multigrafo dirigido y etiquetado
- Soporta *Gremlin Blueprints* (elemento del estándar *TinkerPOP Blueprints)*, un lenguaje para consultar e interactuar con bases de datos de grafos.
- Escrito en java, pero su núcleo está implementado en C^{++}
- Ofrece consultas y opciones de importación con múltiples capacidades de indexación
- Permite a los desarrolladores la visualización de sus modelos de datos para hacer verificaciones, pruebas y modificaciones.

Allegro Graph [22]

Es una base de datos de grafos persistente y de alto rendimiento. Es de código cerrado y está disponible en diferentes versiones (versión gratis y versión empresarial).

Principales características

- Proporciona una arquitectura de protocolo REST, esencialmente un superconjunto de clientes sesame HTTP. Directamente, soporta adaptadores para varios lenguajes como sesame JAVA, sesame JENA, Python utilizando las firmas de sésamo y Lisp. Posee enlaces de código abierto a través de proyectos comunitarios para C #, Ruby, Clojure, Scala, y Perl.
- Tiene indexación automática y dinámica
- Soporta transacciones ACID (Atomicidad, Consistencia, Insolación, Durabilidad)
- Soporta SPARQL, Razonamiento RDFS++[1] y razonamiento con Prolog.

[1] http://www.franz.com/agraph/support/learning/Overview-of-RDFS++.lhtml

- El servidor de Allegro Graph corre nativamente sobre Linux de 32 o 64. Para correr el servidor en otras plataformas se hace necesario de la instalación de una máquina virtual de Linux.

OrientDB [23]

Es un sistema de gestión de base de datos no SQL de código abierto escrito en java. Es escalable y puede trabajar sin esquema, con esquema completo o una mezcla de ambos.

Principales características

- Soporta transacciones ACID, de lenguaje SQL con extensiones para manejar relaciones entre JOINS, manejo de árboles y grafos de documentos relacionados.
- Provee manejo de árboles y grafos 100% compatible con *TinkerPop Blueprints Standard* (colección de interfaces y algoritmos comunes sobre grafos, *Blueprints* es análogo a JDBC para bases de datos relacionales).
- Soporta HTTP, protocolo RESTFUL y JSON.
- Corre sobre cualquier sistema que soporte java.
- Es de pequeño tamaño cuando se usa en modo local, alrededor de un 1mb para el servidor completo. No depende de otros programas y no necesita librerías.
- No soporta SPARQL.

2.1.6 Cuadro comparativo entre las herramientas descritas

La tabla 1, muestra un resumen de características tenidas en cuentas para el desarrollo de la solución de este trabajo

Herramienta	Open source	Soporte SPARQL	Sistema operativo	Lenguaje de programación	Algoritmos sobre grafos
InfiniteGraph	No	No	SUSE Linux, Microsoft, Mac os	Java, C++	Soporta
Neo4j	Si	Si	Multiplataforma	Java	Soporta
AllegroGraph	No	Si	Servidor en Linux, aplicaciones cliente en Windows, mac y linux	Common Lips	No
OrientDB	Si	No	Multiplataforma	Java	Soporta

Tabla 1: Cuadro comparativo entre las herramientas

2.1.7 SPARQL

SPARQL [13] es un acrónimo recursivo de *SPARQL Protocol and RDF Query Language.* SPARQL se puede emplear para expresar consultas sobre fuentes de datos almacenadas nativamente en RDF o vistas como RDF a través de un *middleware.*

La mayoría de las formas de consulta en SPARQL contienen un conjunto de patrones de tripleta (*triple patterns*) denominadas patrón de grafo básico. Los patrones de tripleta son similares a las tripletas RDF, excepto que cada sujeto, predicado y objeto puede ser una variable. Un patrón de grafo básico concuerda con un subgrafo de datos RDF cuando

los términos RDF (*RDF terms*)[2] de dicho subgrafo se pueden sustituir por las variables y el resultado es un grafo RDF equivalente al subgrafo en cuestión.

SPARQL tiene cuatro formas de consulta. Estas formas usan las soluciones de la coincidencia de patrones para formar conjuntos de resultados o grafos RDF. Estas formas son las siguientes:

- **SELECT:** Devuelve tuplas que incluyen todas o un subconjunto de las variables que coindicen con un patrón de consulta.
- **CONSTRUCT:** Devuelve un grafo RDF construido al sustituir las variables en un conjunto de plantillas de tripleta.
- **ASK:** Devuelve un valor booleano que indica si el patrón de consulta existe o no.
- **DESCRIBE:** Devuelve un único grafo RDF con datos sobre recursos.

A continuación se muestra un ejemplo de una consulta en SPARQL, tomada de [13]. La consulta busca encontrar el título de un libro en el grafo de datos dado. Consta de dos partes: la cláusula SELECT donde se identifican las variables que aparecen en los resultados de la consulta y la cláusula WHERE que especifica el patrón de grafo básico que debe concordar con el grafo de datos. El patrón de grafo básico de este ejemplo consiste en un único patrón de tripleta con una sola variable (?title) en la posición del objeto.

Datos:

```
<http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> "SPARQL Tutorial"
```

Consulta:

```
SELECT ?title
WHERE { <http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> ?title }
```

Resultado de la consulta:

title
"SPARQL Tutorial"

2.1.7.1 Modificadores de secuencia de la solución

Los patrones de consulta generan una colección desordena de soluciones. SPARQL provee una serie de modificadores que se aplican sobre el conjunto solución para generar una nueva secuencia de resultados. Los modificadores son los siguientes:

- ***Order By***: Pone las soluciones en un orden específico.
- ***Projection*:** Elige ciertas variables para que aparezcan en la solución.
- ***Distinct:*** Garantiza que las soluciones en la secuencia sean únicas.

[2] http://skos.um.es/TR/rdf-sparql-query/#defn_RDFTerm

- ***Reduced*:** Permite eliminar soluciones no únicas.
- ***Offset*:** Controla dónde comienzan las soluciones en la secuencia global de soluciones.
- ***Limit*:** Restringe el número de soluciones.

2.2 Antecedentes

El proyecto *DISCO* [5], busca integrar información sobre fuentes de datos distribuidas y heterogéneas. Las fuentes de datos pueden ser bases de datos, archivos, servidores de datos dedicados (por ejemplo, un servidor multimedia o una máquina de recuperación de información) o páginas HTML. Así los datos pueden ser estructurados, semi-estructurados o no estructurados. El principal objetivo de *DISCO* es proveer un acceso uniforme y optimizado a las fuentes de datos subyacentes usando un lenguaje común declarativo de consulta.

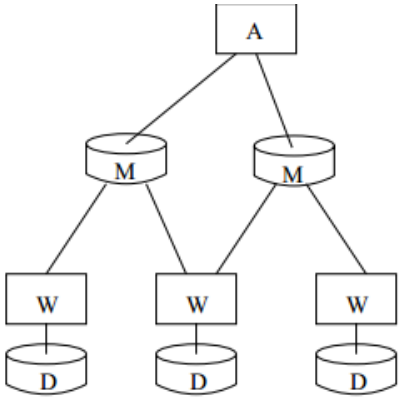


Figura 5: Arquitectura DISCO, fuente [5]

La figura 5 muestra la arquitectura de *DISCO.* 'A' representa las aplicaciones con las que los usuarios finales interactúan. Las aplicaciones acceden a una representación uniforme de las fuentes subyacentes a través de un lenguaje uniforme de consulta declarativa (similar a SQL). 'M' representa a los mediadores que encapsulan una representación de múltiples fuentes de datos para este lenguaje de consulta. Ellos típicamente resuelven los conflictos que involucran la diferencia de representación de conocimiento de diferentes modelos de datos y esquemas y el desequilibrio en el poder de consulta de cada fuente de datos. Esta arquitectura permite desarrollar mediadores independientes y combinarlos, permitiendo un mecanismo que escala de acuerdo con el número de fuentes de datos.

En *DISCO* las subconsultas se expresan en un lenguaje algebraico que soporta operaciones relacionales. 'W' hace referencia a los *wrappers* que dan una vista estructurada de las fuentes de datos y transforman las subconsultas a un particular lenguaje de la fuente de datos. Finalmente, 'D' hace referencia a las fuentes de datos sobre las cuales se realiza la consulta.

La arquitectura de *DISCO* permite optimizar las consultas, ya que para ejecutar el plan de consultas se tiene en cuenta las capacidades de cada fuente, los operadores que soporta y su disponibilidad. Los usuarios expresan las consultas en *DISCO* a través de un lenguaje declarativo muy parecido a SQL y por tanto requiere tener un conocimiento previo.

En la arquitectura para la cual se desarrolla este proyecto de grado, los usuarios pueden expresar sus consultas en un lenguaje conceptual, cercano al usuario, en este caso orientado al campo médico, sin tener que preocuparse por expresar sus consultas en un lenguaje especifico como el utilizado en *DISCO*. Además, no están limitados al lenguaje, se tienen una mayor expresividad y saca ventaja del conocimiento del usuario experto en el tema de la consulta.

Un aspecto a resaltar de *DISCO* es que nuevas fuentes se pueden añadir fácilmente. Cuando una nueva fuente se añade, el *wrapper* de esa fuente se encarga de enviarle al mediador las operaciones que soporta y su costo. El mediador registra estos datos y los tiene en cuenta para elaborar los planes de consulta. En comparación con la arquitectura propuesta esta actualización o inclusión de una fuente no se hace de forma automática.

TUKWILA [6], es un sistema de integración de datos que trata de responder a las consultas realizadas a diferentes fuentes de datos autónomas y heterogéneas. Todas las fuentes se trasladan a un esquema intermedio común. El sistema de integración de datos reformula las consultas en varias consultas sobre las fuentes y posteriormente combina los resultados en una sola respuesta. El objetivo de *TUKWILA* es procesar consultas eficientemente sobre cadenas de datos en XML. La figura 6 muestra la arquitectura de *TUKWILA.*

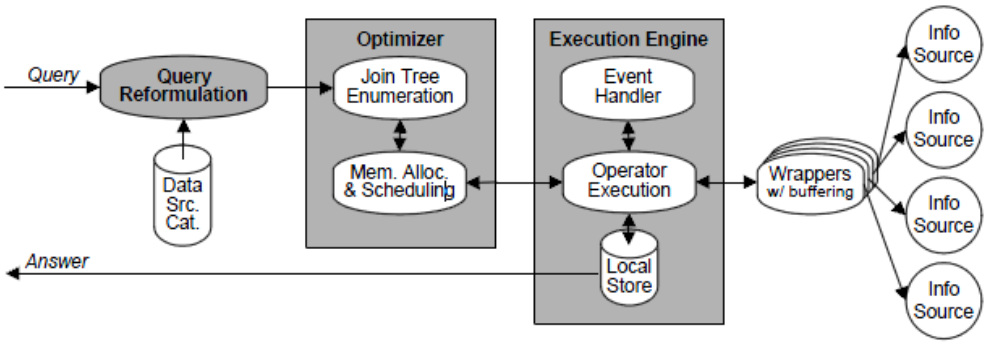


Figura 6: Arquitectura TUKWILA, fuente [6]

Las consultas en *TUKWILA* se plantean en términos de un esquema mediado relacional. El objetivo del esquema mediado es abstraer los detalles de las fuentes de datos. El *data source catalog* contiene varios tipos de metadatos sobre las fuente de datos. Se puede encontrar una descripción semántica de los contenidos de las fuentes de datos, información acerca de pares de fuentes de datos (fuentes de datos en las que aparecen

un mismo valor de datos) y estadísticas clave sobre los datos tales como: el costo de acceder a cada fuente de datos, los tamaños de las relaciones en las fuentes e información sobre la selectividad. En la *query reformulation*, la consulta sobre el esquema mediado se convierte en una unión de consultas conjuntivas a partir de los esquemas de las fuentes de datos. El *optimizer* transforma la consulta en un plan de ejecución de consultas para el *execution engine*. El optimizador tiene la capacidad de crear planes parciales. Si las estadísticas esenciales no existen o son inciertas y tiene reglas para un comportamiento adaptativo en tiempo de ejecución. El *execution engine* procesa los planes de consulta producidos por el optimizador. Incluye un controlador de eventos para la interpretación dinámica de las reglas y soporta reoptimizacion incremental. Por último, los *wrappers*, manejan la comunicación con las fuentes de datos y, cuando es necesario, traducen los datos del formato usado en *TUKWILA* a los formatos utilizados en las fuentes.

TUKWILA posee una gran fortaleza en la optimización y re-optimización de los planes de consulta, que buscan el tiempo de ejecución de la consulta y adaptar el plan durante su ejecución como respuesta a situaciones del contexto, por ejemplo cuando la memoria es insuficiente.

Con respecto a la arquitectura para la cual se desarrolla este proyecto de grado, en *TUKWILA* no se tiene un elemento marcado como mediador pero se puede decir que este elemento estaría enmarcado por las funciones que se realizan en la parte del optimizador y en la parte de máquina de ejecución de las consultas.

Por la inclusión de la adaptabilidad en las consultas se puede evidenciar que TUKWILA se enfoca en la optimización y en la ejecución para lograr un mayor rendimiento en la integración de los datos.

TSIMMIS [7], es un proyecto conjunto entre Stanford e *IBM Almaden Research Center*, cuyo objetivo es desarrollar herramientas que faciliten la integración rápida de fuentes de información heterogéneas que pueden incluir datos estructurados y semiestructurados. Sus componentes pueden traducir preguntas e información, extraer datos de la Web y combinar información de diversas fuentes vía un mediador.

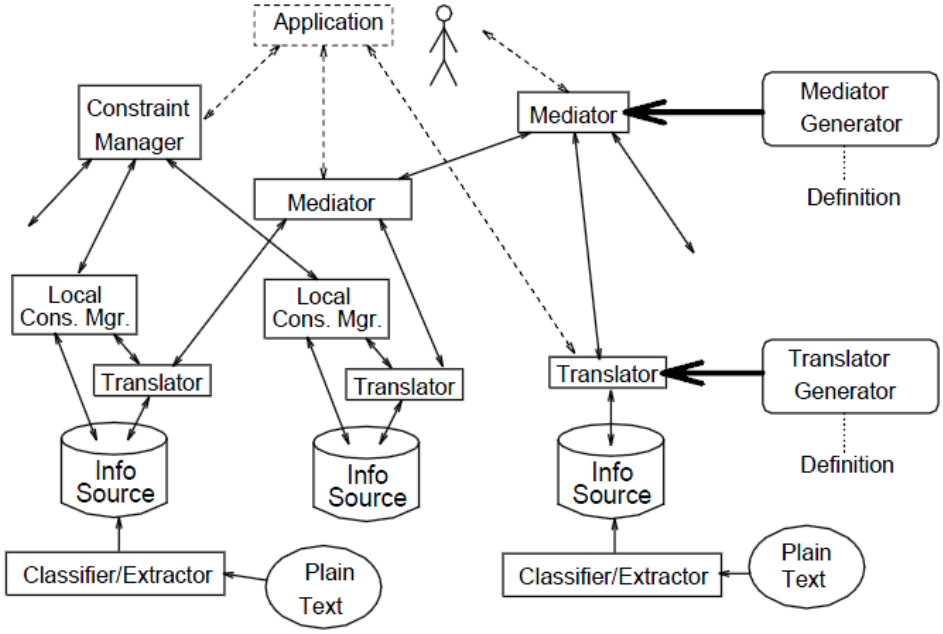


Figura 7: Arquitectura TSIMMIS, fuente [7]

En la figura 7 se muestra la arquitectura de TSIMMIS. Un conjunto de discos representan las fuentes de datos heterogéneas. Por encima de cada fuente se tienen *translators* (*wrappers*), que mapean los objetos subyacentes de datos a un modelo de información común. Los *translators* convierten las consultas expresadas según el modelo común en solicitudes que las fuentes puedan responder, y convierten los datos retornados a la representación unificada de la información. *TSIMMIS* adopta un modelo simple de objeto etiquetado llamado *OEM* (*Object Exchange Model*). OEM permite la anidación de objetos. Todos los objetos y sub-objetos tienen etiquetas que describen su significado. El siguiente es un ejemplo de un objeto que representa una temperatura en Fahrenheit de 80 grados:

(temp-in-Fahrenheit, int, 8)

donde la cadena temp-in-Fahrenheit representa la etiqueta, int indica un valor entero y 80 representa el valor del objeto definido.

OEM es simple, flexible y provee la expresividad necesaria para integrar información de fuentes heterogéneas. OEM-QL es el lenguaje de consulta desarrollado para la solicitud de objetos OEM en *TSIMMIS*. OEM-QL es un lenguaje muy similar a SQL, extendido para tratar las etiquetas y los objetos anidados.

En la figura 7, por encima de los *translators se* encuentran los *mediators* (mediadores). Un mediador integra el conocimiento que es necesario para procesar un tipo específico de información. *TSIMMIS* se enfoca en la utilización de mediadores simples basados en patrones y reglas.

Implementar un mediador puede ser complicado y puede consumir mucho tiempo por lo que uno de los objetivos del proyecto es, automáticamente o semiautomáticamente,

generar mediadores que se ajusten a las necesidades de la capa de aplicaciones. En la figura 7 se muestra un *mediator generator* y un *translator generator*, estos últimos generan *OEM translator* con base en una descripción de conversiones que toman lugar cuando se reciben las consultas y se retornan los resultados.

Otro componente de la arquitectura es el *Constraint Manager* (gestores de restricción). Este componente apoya la definición de interfaces que soporta una fuente para la información involucrada como restricción (por ejemplo, puede un *trigger* fijarse como un elemento de datos), la especificación de la restricción deseada (por ejemplo, dos elementos que tenga el mismo valor) y la especificación de estrategias que deben ser seguidas cuando se cumpla una restricción o se detecten violaciones.

El último componente de la arquitectura de la figura 7 son los *Classifier/Extractor* (clasificadores/extractores). Son encargados de extraer la información de fuentes no estructuradas en las cuales a menudo es complicado clasificar automáticamente los objetos (por ejemplo, un archivo plano o un mensaje de correo).

Un punto a resaltar en *TSIMMIS* es que las peticiones a los mediadores y a los traductores se realizan en un mismo lenguaje: *OEM-QL*. Estos dos componentes retornan la información representada en objetos *OEM*. Esta unificación de representación por medio de objetos *OEM* permite a los usuarios obtener información tanto de los mediadores como de los traductores y a los mediadores acceder transparentemente a nuevas fuentes de datos.

Una ventaja con respecto a la arquitectura para la cual se desarrolla este proyecto de grado es que la representación del modelo común (*OEM*) que utiliza *TSIMMIS* permite que nuevas fuentes se puedan añadir y que las fuentes actuales puedan manejar información dinámica. También, cabe mencionar, que *TSIMMIS* brinda acceso a fuentes no estructuradas o semi-estructuradas y en ocasiones a fuentes que no tengan un esquema regular que las defina.

El proyecto MIDAS [8] nace de la colaboración entre el grupo de informática médica del BET de la Universidad Politécnica de Valencia y el Servicio de informática del Hospital la Fe. Intenta dar solución a la demanda, por parte del personal sanitario, de una aplicación que les permita tener acceso a una vista unificada de las historias clínicas de los pacientes, con el fin de agilizar algunos de los procesos que tienen los diferentes departamentos del hospital, como el acceso a los resultados de las pruebas de laboratorio o al historial de pruebas radiológicas que se le han realizado a un determinado paciente.

MIDAS, como sistema de integración de datos, sigue una arquitectura basada en mediadores y adaptadores. En la figura 8 se muestra la arquitectura general de MIDAS.

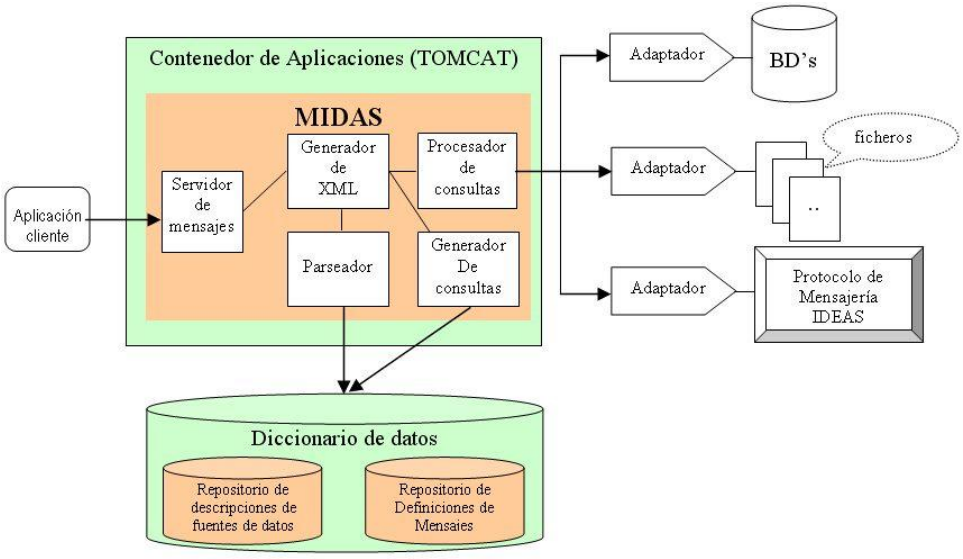


Figura 8: Arquitectura MIDAS, fuente [8]

MIDAS se encarga de atender cualquier petición clínica-administrativa sobre los pacientes y por lo tanto, se sitúa entre los usuarios finales y los repositorios de datos. La comunicación entre las aplicaciones cliente está basada en el uso de un conjunto de servicios ofertados por el servidor.

Los usuarios finales reciben la información de los pacientes por medio de un mensaje. Este mensaje contiene un documento XML con la información solicitada. Los mensajes se definen en un lenguaje basado en XML llamado MAD. Con este lenguaje es posible especificar fuentes de datos con la información necesaria, información que se va a extraer de esas fuentes, etiquetado y estructura del documento XML resultante, parámetros que aceptan el mensaje y las combinaciones validas de estos.

El sistema hace uso de un diccionario de datos, como se muestra en la figura 8, que contiene la descripción de los mensajes y la descripción de las fuentes conectadas al sistema. Además cuenta con cuatro componentes básicos: el generador de XML, el procesador de consultas, el generador de consultas y el parseador. El parseador analiza los mensajes disponibles en el diccionario de datos y crea una estructura interna (árbol) que simula la estructura del mensaje respuesta, en la que cada nodo se corresponde con un elemento XML simple o compuesto. Si es un elemento simple, el nodo correspondiente es una hoja del árbol y tiene asociada una expresión que involucra información procedente de las fuentes de datos. Un elemento compuesto, únicamente define cómo estructurar los datos en el mensaje respuesta y puede tener asociada una plantilla de consulta para obtener la información a la que hacen referencia sus nodos descendientes. Cuando el usuario del sistema realiza una petición de instanciación de uno de los mensajes para un caso concreto (e.g. datos demográficos de un paciente o resultados de un análisis), el generador de XML interpreta el árbol construido previamente y haciendo uso del generador de consultas crea las consultas necesarias para obtener la información

requerida. Estas consultas las envía y gestiona el procesador de consultas. Una vez obtenidos los datos el generador de XML construye con ellos el mensaje respuesta que se envía finalmente al cliente.

Una gran ventaja que se presenta MIDAS es que gracias a que usa tecnologías abiertas, como XML, para definición de los mensajes, las fuentes y la comunicación de la información, diferentes aplicaciones que se ejecutan en diferentes sistemas operativos y arquitecturas pueden acceder al sistema. Además, estas tecnologías están teniendo una gran difusión y existen gran cantidad de librerías y utilidades para su uso en la mayoría de los entornos de desarrollo existentes.

Con respecto a la arquitectura para la cual se desarrolla este proyecto de grado, se puede ver que el elemento mediador está implícito en la parte del contenedor de aplicaciones y el diccionario de los datos. En esta parte se realiza la misma función que en la arquitectura propuesta pero con la diferencia que la mediación no está enmarcada sobre un lenguaje conceptual. La información se entrega a las aplicaciones en mensajes XML y estos se encargan de mostrarla a los usuarios.

Hernández Cardona [14] proponen como solución a un problema de integración que permite el acceso a dos conjuntos de datos distintos de egresados de instituciones de educación superior de México. Se trata de dos bases de datos relacionales. La primera, corresponde al esquema de la base de datos del sistema SIBOLTRA del Departamento de Egresados y Servicio Social del Instituto Politécnico Nacional, aunque con acceso solo a los datos de los egresados la Unidad Profesional Interdisciplinaria de Ingeniería y Tecnologías Avanzadas (UPIITA). La segunda, corresponde a los datos de los egresados de la Escuela Superior de Cómputo (ESCOM). En ambos casos los datos se toman del SIBOLTRA pero están almacenados bajo diferentes esquemas. El acceso unificado a estos conjuntos de datos permite responder de forma unificada a preguntas como: ¿Quiénes son los egresados de cierta carrera? ¿Quiénes de ellos viven más cerca de una colonia o población?, ¿Quiénes tienen edad bajo cierto rango?, entre otras.

De relevancia significativa para este trabajo son los datos relativos a la ubicación geográfica del domicilio de residencia del egresado. Las bases de datos relacionales de los egresados de la UPITA y ESCOM se consideran fuentes de datos espaciales, ya que contienen información relativa a la ubicación geográfica del domicilio de residencia del egresado. Esta información se utiliza a través del código postal y usa la API de Google Maps, para mostrar en un mapa, en una página web, la localización de los profesionales egresados de acuerdo con las consultas generadas con base en estas preguntas.

La principal diferencia entre los conjuntos de datos utilizados se refiere a la conceptualización del dominio acerca del cual guardan información, es decir de los egresados de la institución. Las diferencias en la conceptualización del dominio redunda en diferencias en el esquema de base de datos y en los términos que se emplean para definirlo, ya que cada diseñador de la base de datos hace la abstracción del dominio de acuerdo a la realidad que lo circunscribe, la cual evidentemente puede ser distinta y para ello emplea términos de lenguaje distintos.

La figura 9, muestra la arquitectura de la solución propuesta [14].

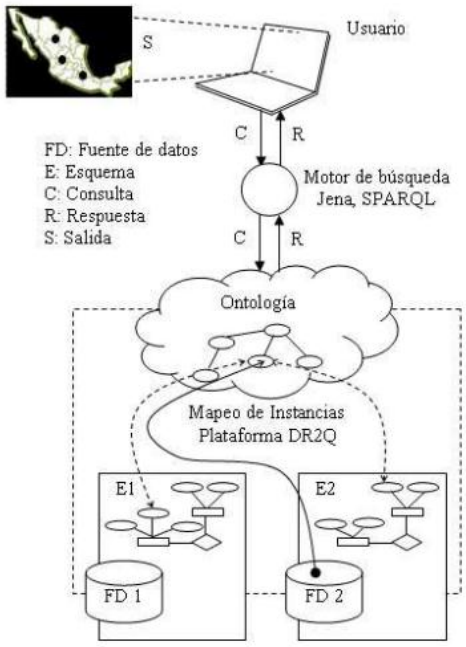


Figura 9: Arquitectura de la solución propuesta, fuente [14]

La solución propone el uso de una ontología como una capa intermedia entre la aplicación o interfaz para el manejo de la información y las fuentes de datos. Cada uno de los elementos de cada esquema seleccionado para ser integrado se mapea a una clase o elemento dentro de la ontología. Con el uso de la ontología, los elementos de los esquemas no necesitan guardar una relación explicita entre sí o entrelazarse unos con otros al momento de realizar una consulta. La consulta se construye en términos de la ontología, usando el lenguaje SPARQL. Para mapear los elementos de los esquemas de las bases de datos con las clases definidas en la ontología, se crea un modelo RDF a partir de un archivo de texto donde se establecen explícitamente estas asociaciones. La herramienta utilizada para la creación de modelo RDF es D2RQ, la cual tiene el soporte de API de programación de Jena, que permite el procesamiento de las consultas en SPARQL a través del modelo creado

El usuario realiza la búsqueda de información sobre ciertos campos de la aplicación, que le permiten establecer criterios para la recuperación de la misma. Con estos criterios se envía una consulta SPARQL. La herramienta D2RQ reescribe las consultas SPARQL en consultas SQL especificas del modelo de datos de la aplicación. El conjunto de resultados de estas consultas SQL se transforman en tripletas RDF que se entregan a las capas superiores de framework de Jena para luego mostrar los resultados al usuario.

En esta arquitectura se puede notar que el componente mediador se encuentra implícito entre la ontología y las fuentes de datos. La herramienta D2RQ prácticamente es la

encargada de realizar la mediación, ya que permite el acceso unificado a las fuentes de datos. En este trabajo no se hizo necesaria la utilización de *wrappers* en las fuentes de datos ya que esta tarea es realizada también por D2RQ.

Botello [15], se enfoca en el desarrollo de un sistema de integración que permite establecer conexiones con bases de datos remotas y heterogéneas para extraer sus esquemas constitutivos. Además, hace inferencias del contenido de los datos de cada fuente de información mediante el uso de algoritmos de concatenación para obtener una estructura común, y resolver consultas globales de usuario usando una interfaz gráfica que reúna diferentes y diversas fuentes de información.

Este sistema es capaz de resolver consultas como "la edad promedio de los estudiantes de nivel superior que viven en el norte de la Ciudad de México y que sus padres trabajen en alguna oficina gubernamental". Para la solución de consultas como esta se propone como primer paso el uso de una ontología con los términos más comúnmente empleados en la búsqueda de información, tales como los encontrados en oficinas del gobierno, instituciones académicas y de investigación, empresas comerciales, depositarios públicos, etc. Esta ontología permite al usuario guiar su búsqueda incluyendo sólo las fuentes información que ofrezcan resultados parciales. Mediante el desarrollo de una interfaz gráfica, en donde cada fuente de datos se muestra como un grafo conexo, el usuario puede dibujar líneas de conexión entre cada fuente de información para establecer las condiciones a ser aplicadas. En un segundo paso, se hace la transformación del grafo en consultas que deben ser resueltas por el sistema de bases de datos que involucra a la fuente de datos respectivo, escritas en SQL. En esta fase se infieren las relaciones entre cada fuente de información que participa en la consulta global, esto es, si el sitio que tiene la edad promedio de los estudiantes no incluye la orientación geográfica de donde viven ellos, se necesita encontrar algún tipo de relación con otra fuente de datos no considerada inicialmente, como podrían ser los mapas cartográficos de la localización de escuelas en la Ciudad de México. Después de encontrar estas relaciones, se procede a resolver las inconsistencias (si existen) en los datos obtenidos como resultado parcial (los nombres de estudiantes como Juan Pérez o Pérez, Juan), empleando algoritmos de inteligencia artificial para unificar los conceptos empatados. Finalmente, se hace la correspondencia para cada elemento de cada fuente de datos a fin de que sea consistente con el esquema común obtenido, y se integran los resultados de cada fuente de información para que sean mostrados al usuario.

En este trabajo la mediación ocurre cuando la trayectoria trazada por el usuario sobre la interfaz gráfica se transforma en consultas SQL sobre las bases de datos, que luego se integran para obtener una respuesta total y entregarla al usuario.

El proyecto virtuoso [16], dentro de su portafolio de funcionalidades, tiene una parte dedicada la integración de datos. Virtuoso dentro de su núcleo contiene un motor de base de datos virtual que trabaja con los estándares de la industria y tecnologías de acceso de datos para proveer acceso transparente a fuentes de datos dispersas como SQL, XML, texto libre, entre otras.

El motor de base de datos virtual de virtuoso oculta las diferencias de dialectos de las bases de datos externas. Las aplicaciones, los datos y los desarrolladores ven unificada la sintaxis de consulta y la estructura. Las consultas y los procedimientos almacenados pueden abarcar un número arbitrario de bases de datos.

El modelo unificado en el motor de base de datos virtual está disponible para cualquier aplicación a través de cualquier punto de acceso comunicándose por un solo dialecto, en este caso SQL. Este motor ofrece una gama completa de apis cliente, incluyendo ODBC, JDBC, OLE/DB y adaptadores de datos .net. Las rutas de consulta son optimizadas automáticamente y se provee un único punto de administración de la seguridad y los privilegios de acceso al subyacente mundo de los datos corporativos.

Informática dentro de su gama de productos, ofrece un software para integración virtual llamado "*Data Virtualization*" [17], que proporciona a las empresas una capa de abstracción de datos para obtener un acceso rápido y directo a los datos de muchas fuentes diversas sin necesidad de moverlos físicamente. Usan una arquitectura denominada "*Informatica Data Services*", que permite la creación de una capa de virtualización de datos que oculta y gestiona la complejidad necesaria para acceder a las fuentes de datos subyacentes, aislándolos a su vez de los cambios. Como resultado con esta arquitectura, los analistas obtienen los datos y la confianza que necesitan, mientras que la organización IT conserva el control del proceso.

La figura 10, muestra la arquitectura "*Informatica Data Services*"

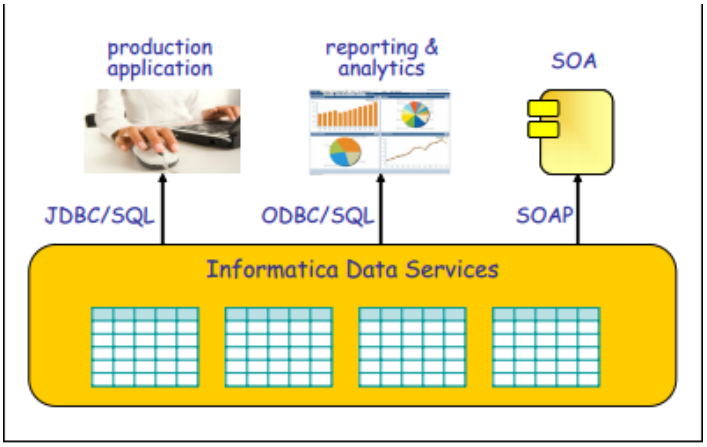


Figura 10: Arquitectura Informatica Data Services, fuente [17]

Esta arquitectura muestra todos los datos disponibles a través de una capa que contiene tablas llamadas "tablas virtuales" que consisten de filas y columnas. Esas tablas se pueden acceder por medio de interfaces SQL o a través interfaces basadas en SOAP. Una tabla virtual no almacena datos, su contenido es solamente virtual. Contiene la descripción de todas sus columnas, sus respectivos tipos de datos y relaciones con otras tablas virtuales. Antes de que una tabla virtual se use, ésta primero tiene que enlazar a

una o más fuente de datos (XML, archivos planos, bases de datos SQL) que se representan en esta arquitectura como “tablas foráneas”. Este enlace se realiza en dos pasos. En el primer paso, se crea un objeto de datos físico que contiene información de la estructura de la fuente real de datos más especificaciones de dónde y cómo están almacenados los datos de las fuentes reales y cómo acceder a estos. El cierto sentido, el objeto de datos físico se puede ver como un *wrapper* alrededor de una tabla foránea.

En el segundo paso, la tabla virtual se enlaza con uno o más objetos de datos físicos. Esto se realiza a través de mapeos entre objetos de datos. El mapeo define cómo los datos de esos objetos físicos subyacentes se deberían mapear para llegar a ser contenidos virtuales de una tabla virtual. Si una tabla virtual no está enlazada a un objeto de datos físico, esta no podrá ser consultada.

La figura 11, muestra las relaciones entre los conceptos de tabla virtual, mapeo de objeto de datos y objeto físico de datos.

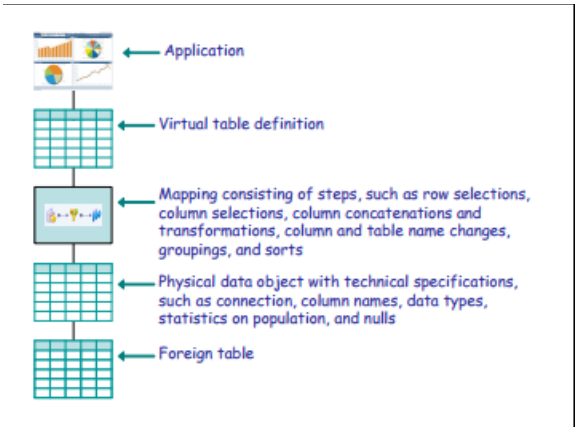


Figura 11: Relaciones entre los conceptos, fuente [17]

Con la creación de un objeto de datos físico, una tabla foránea llega a ser conocida en “*Informatica Data Services*” y por lo tanto llega a ser accesible por las diferentes aplicaciones. El proceso de crear un objeto de datos físico se referencia como “importar tabla foránea” y es relativamente fácil. *Informatica Data Services* crea una conexión con la fuente de datos requerida y recupera los metadatos en tablas del catálogo, con nombres de columnas, tipos de datos, especificaciones nulas y población de datos. Estos datos se almacenan en el catálogo de *Informatica Data Services*. Una vez que una tabla foránea se ha importado, un objeto de datos físico se crea, los mapeos se podrán definir y el objeto se puede hacer disponible a las tablas virtuales.

La definición de los mapeos entre las tablas virtuales y los objetos de datos físicos se realiza por medio de un lenguaje gráfico. Estos son traducidos a consultas SQL que procesa un motor SQL que hace parte de la arquitectura. El motor SQL permite recuperar datos de diferentes fuentes ejecutando una sentencia SQL. El lenguaje SQL está oculto a

los desarrolladores, ellos solo trabajan en la definición de los mapeos con el lenguaje grafico que proporciona "*Data Virtualization*".

En este trabajo el modelo de datos se encuentra representado por las tablas virtuales que describen los datos disponibles en las fuentes de datos. Los desarrolladores describen las tablas virtuales al igual que los mapeos, por medio de herramientas de la aplicación. La mediación se realiza en el momento que una consulta relaciona una tabla virtual, esta tendrá relacionada un mapeo a un objeto de datos físicos que se pueden ver como *wrappers* sobre las fuentes. Este mapeo se ejecuta sobre las fuentes como una consulta SQL para recuperar los datos y entregarlos al usuario.

Microsoft Corporation ofrece una plataforma para integrar datos procedentes de diferentes fuentes de datos llamada ADO.NET *Entity Framework* [18]. Es una herramienta para programación que eleva el nivel de abstracción del nivel lógico (relacional) al nivel conceptual (entidad), y por lo tanto reduce la desigualdad de impedancia para aplicaciones y servicios de datos tales como informes, análisis y replicación. El modelo conceptual es una extensión del modelo relacional llamado *The Entity Data Model* o EDM, que abarca las entidades y las relaciones como conceptos de primera clase; posee un lenguaje de consulta para el EDM, un motor de mapeo que traduce del nivel conceptual al lógico y un conjunto de herramientas que ayudan a creación de transformadores para objetos entidad, objetos XML y entidades XML.

El EDM extiende el modelo relacional clásico con conceptos de modelado E-R. Los conceptos centrales del EDM son las entidades y las relaciones. Las entidades representan objetos del nivel superior con existencia independiente e identidad, mientras que las relaciones se usan para relacionar dos o más entidades. El modelo se puede definir explícitamente por medio de XML o alternativamente usando una herramienta grafica para hacerlo.

ADO.NET *Entity Framework* soporta una innovadora arquitectura de mapeos que contiene las siguientes características:

-Especificación: Los mapeos se especifican utilizando un lenguaje declarativo que tiene una semántica bien definida y ofrece una amplia gama de escenarios de mapeos a los usuarios no expertos.

-Compilación: Los mapeos se compilan en vistas bidireccionales, llamadas vistas de consulta y vista de actualización, que se manejan y procesan por un componente de la arquitectura llamado "máquina de ejecución".

-Ejecución: La actualización se realiza usando un mecanismo general que aprovecha el mantenimiento de vistas materializadas, una robusta tecnología de bases de datos. La consulta se realiza usando el enfoque GAV.

Los mapeos se especifican usando un conjunto de fragmentos de mapeos. Un fragmento de mapeo describe cómo una porción de un EDM corresponde a una porción del esquema de una base de datos (modelo relacional).

La figura 12, muestra la arquitectura de ADO.NET *Entity Framework* y a continuación de describen sus componentes principales:

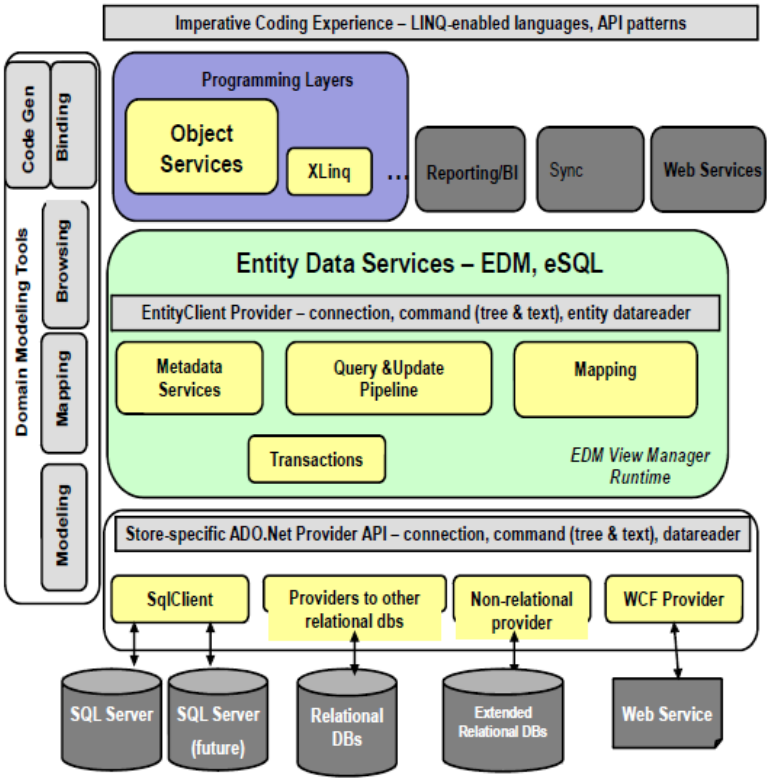


Figura 12: Arquitectura ADO.NET Entity Framework, fuente [18]

Fuentes de datos y proveedores específicos: ADO.NET *Entity Framework* se basa en el modelo *data provider.* Los proveedores son componentes que establecen la comunicación con una fuente específica. Hay proveedores específicos para esquemas relacionales, no relacionales, y fuentes de servicios web.

Proveedor *EntityClient*: Representa una capa conceptual de programación. Los datos se acceden por medio de entidades EDM y se consultan y actualizan usando un lenguaje basado en SQL llamado *Entity* SQL. Incluye también una vista del subsistema de mapeos que soporta actualización de vistas EDM alrededor de tablas relacionales.

Servicios de objetos y otras capas de programación: Provee una rica abstracción de objetos a través de entidades, un rico conjunto de servicios alrededor de esos objetos y permite a las aplicaciones programar usando construcciones familiares de lenguajes de programación.

El *Entity Framework* permite que múltiples capas de programación puedan ser conectadas a la capa de *Entity Data Services* expuesta por el Proveedor *EntityClient.*

Servicios de metadatos: Componente que maneja metadatos para el tiempo de diseño y ejecución requeridos en *Entity Framework* y las aplicaciones alrededor de este. Los metadatos manejados incluyen los asociados con conceptos EDM (entidades, relaciones,

asociaciones, etc.), conceptos de almacenamiento (tablas, columnas, restricciones) y conceptos de mapeo.

Herramientas de diseño y metadatos: *Entity framework* integra diferentes herramientas para los diseñadores que facilitan el desarrollo de aplicaciones. Dentro de este paquete de integración se encuentran herramientas para el diseño de esquemas EDM, diseño de mapeos, herramientas de generación de código, entre otras.

Servicios: Servicios tales como reportes, sincronización, análisis de negocio pueden ser construidos con el framework.

En este trabajo la mediación es hecha por la capa de *Entity Data Services,* encargada de recibir las solicitudes de diferentes aplicaciones y de retornar los resultados. Contiene el modelo conceptual EDM y las equivalencias entre porciones del modelo conceptual y los esquemas lógicos de las fuentes adyacentes (mapeos). Los proveedores, que permiten establecer la conexión con las fuentes de datos, se pueden ver como *wrappers* hacia ellas.

2.3 Marco Conceptual

Modelo de datos conceptual

Los modelos de datos conceptuales se caracterizan por permitir crear representaciones simplificadas de objetos, fenómenos o situaciones reales, ser precisos, completos, expresivos y fáciles de usar, posibilitando describir, a alto nivel, aspectos relevantes de un dominio en particular, usando un vocabulario propio, con términos y conceptos familiares al usuario [1]. Algunos de los modelos conceptuales más conocidos son el modelo entidad relación y ORM.

XML *(Extensible Markup Language)*

Es un lenguaje universal, extensible, que permite la compatibilidad entre todo tipo de dispositivos y programas. Se dice que es extensible ya que en XML se pueden definir etiquetas que demarcan, por su nombre, la semántica de los datos que encapsulan. De esta manera, cualquier aplicación que conozca las etiquetas usadas en un documento XML concreto, puede entender el contenido del mismo. El lenguaje de marcas que emplea, le da a los datos descritos un significado semi-semántico, pero las marcas en sí mismas sólo tienen carácter sintáctico [9,10].

Consultas *ad hoc*

Son consultas específicas y personalizas creadas por los mismos usuarios, a través de sistemas basados en interfaces amigables sin necesidad de tener conocimiento a fondo de SQL o del esquema de la base de datos que el programador ha desarrollado [11].

Capítulo 3

En este capítulo se presenta el diseño que se plantea como solución al problema. El capítulo está organizado en nueve secciones. En la primera sección, se presenta la metodología de desarrollo utilizada para el proyecto. En la segunda sección se describen sus principales características. En la tercera sección se muestran cada una de sus fases y los entregables de estas para el proyecto. En la cuarta sección se describe el modelo de grafos usado para representar el modelo global del mediador. En la quinta sección se justifica la selección de la herramienta escogida para su implementación. En la sexta sección, se describe cómo está representado el modelo global en la herramienta. En la séptima sección se describe cómo se almacenan los mapeos. En la octava sección se muestra que información se guarda de las fuentes. Por último, se presenta la evaluación de diferentes formas de implementación para el modelo global.

3.1 Metodología

La metodología de desarrollo de software escogida para implementar la solución propuesta fue SCRUM, ya que al ser una metodología ágil se adapta a las características de desarrollo del proyecto, teniendo en cuenta que se requería probar nuevas tecnologías (capacidad de respuesta a cambios) además de la entrega continua en plazos breves de software funcional y de mantener un contacto permanente entre el cliente y el equipo de desarrollo. Se escogió trabajar con esta metodología ágil y no en otra comúnmente utilizada como XP (*eXtreme Programing*), debido a que una de las prácticas de XP sugiere que la codificación se debe lleve a cabo en parejas, algo con lo que no se contaba para este proyecto.

Para facilitar el proceso de desarrollo y definir claramente el alcance de la aplicación, se decidió complementar la metodología SCRUM haciendo uso de algunos artefactos de la metodología RUP (*Rational Unified Process*) [24]. RUP cuenta con una especificación completa y clara de requerimientos que se usó en la etapa de pre juego (planeación) descrita en SCRUM. Una descripción más detallada de SCRUM y los resultados de sus fases serán presentadas en la siguiente sección.

3.2 Descripción de la Metodología de Desarrollo de Software

SCRUM es actualmente una de las metodologías ágiles para el desarrollo de software de mayor difusión en la industria, junto con XP. Es una metodología iterativa e incremental que enfatiza prácticas y valores del *Project management* por sobre las demás disciplinas del desarrollo. Al principio del proyecto se define el *Product Backlog,* que contiene los requerimientos funcionales y no funcionales con los que deberá contar el sistema a

construir. Estos estarán especificados de acuerdo las convenciones propias de cada organización, aunque usualmente se utiliza un artefacto de alto nivel, cercano al usuario como lo son las historias de usuario para describirlos. El *Product Backlog* se define durante reuniones con los interesados en el proyecto (*Stakeholders*). A partir de ahí se definen las iteraciones, conocidas como *Sprint* en la juerga de SCRUM, en la que se ira construyendo la aplicación evolutivamente. Cada *Sprint* tendrá su propio *Product Backlog* que será un subconjunto del *Product Backlog* definido inicialmente con los requerimientos a ser construidos en el *Sprint* correspondiente. La duración recomendada para cada *Sprint* es de una hasta 4 semanas [25].

La figura 13 describe el proceso de desarrollo de SCRUM:

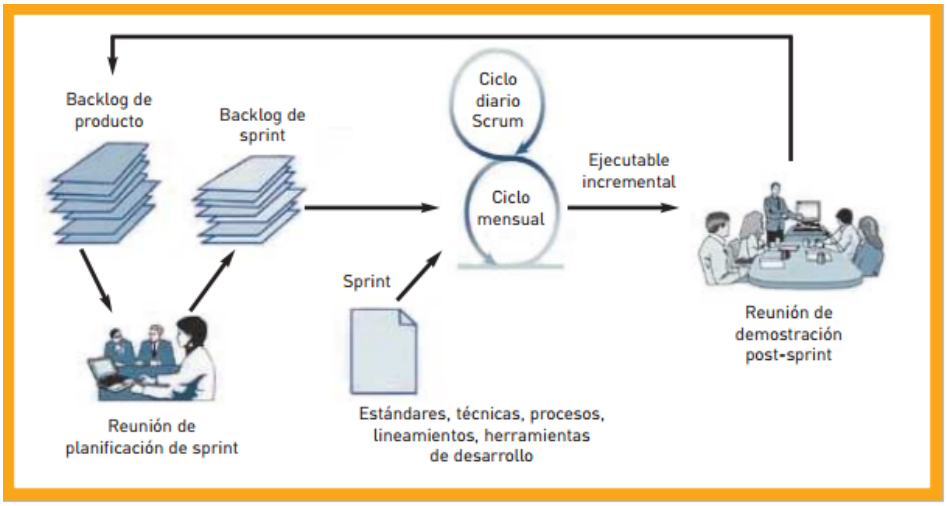


Figura 13: Descripción proceso de SCRUM, fuente [26].

3.3 Fases de SCRUM

SCRUM describe el proceso de desarrollo en 3 fases: pre-juego (planeación y montaje), juego (desarrollo) y pos-juego (liberación) [25].

3.3.1 Pre-juego (planeación y montaje)

La fase de pre-juego encierra la planeación y la arquitectura. En la planeación se tiene como artefacto principal el *Product Backlog.* En este se presentan los requisitos del sistema en un lenguaje no técnico, descritos en historias de usuario generalmente. Estos requisitos son priorizados por valor de negocio por el cliente y son revisados en intervalos regulares dentro del proceso de desarrollo. Para el proyecto se elaboraron historias que describen las funcionalidades del sistema de forma modular. Cada módulo y sus historias de usuario fueron definidos de acuerdo a las necesidades planteadas en los requerimientos. Los requisitos del *Product Backlog* fueron priorizados de acuerdo a la complejidad que implicaba su desarrollo. Los requisitos más complejos contemplan una prioridad menor y los requisitos más básicos contemplan una prioridad mayor. Esta

priorización permitió que las iteraciones que abarcaban requisitos básicos soportaran las iteraciones de requisitos más complejos.

3.3.1.1 Especificación de requerimientos

Para apoyar las historias de usuario y para dar un mayor detalle del alcance del sistema a construir, se definió primeramente una especificación de requerimientos. Con la especificación de las necesidades en los requerimientos se delimitó el alcance del sistema y se obtuvo una definición concreta de las funcionalidades para dar cumplimiento a los objetivos propuestos. En la tabla 2 se muestran los requerimientos para el módulo de consultas. Por características del proyecto no se tiene una interacción directa con el usuario. Se recibe una entrada (consulta) proveniente de la interfaz de usuario y el sistema como salida retorna un conjunto de subconsultas a las fuentes de datos.

	Especificación de Requerimientos ISO-9000 Universidad del Valle	**Documento:** ER-001	**Revisión:** 001
Especificación de Requerimientos de la aplicación		**Página:** 14/25 **Fecha:** 21/03/2013	

1. Módulo: Consultas.

Req.	Descripción
R1.1	El sistema en el módulo de consultas, recibe la consulta proveniente de una interfaz de usuario e identifica qué tipo de operador involucra la consulta y que parámetros contiene.
R1.2	El sistema en el módulo de consultas, debe identificar cuáles nodos se retornan como respuesta y cuáles arcos forman parte del patrón de búsqueda dependiendo del tipo de operador definido en la consulta (filtro y subgrafo).
R1.3	El sistema en el módulo de consultas, debe identificar cuáles nodos tienen un filtro si el operador de la consulta es de tipo filtro.
R1.4	El sistema en el módulo de consultas, debe identificar cuáles fuentes de datos están involucradas en la consulta de un usuario.
R1.5	El sistema en el módulo de consultas, debe dividir la consulta realizada en subconsultas que cada fuente está en capacidad de responder
R1.6	El sistema en el módulo de consultas, debe reformular las subconsultas para que las entidades del modelo global se mapeen a entidades equivalentes definidas en el modelo local de la fuente de datos.

Tabla 2: Requerimientos, módulo de consultas.

El documento completo de especificación de requerimientos se encuentra en el Anexo A.

3.3.1.2 Product Backlog

Es el corazón de SCRUM, es una lista priorizada de requisitos funcionales usualmente descritos en historias de usuario. Para el proyecto, en el *Product Backlog* se identifica cada característica básica a implementar, titulo, descripción, la prioridad definida entre 1 y 10 y los criterios de aceptación para corroborar que la funcionalidad descrita se supla. La tabla 3 muestra el *Product Backlog* del módulo de consultas.

	*Product Backlog*	Documento: PB-001	Revision:001	Fecha: 21/03/2013
Modulo: Consultas				
ID	Título	Descripción	Prueba de Aceptación	Prioridad
H01	COMO usuario quiero que el sistema identifique que fuentes están involucradas en una consulta	Al sistema llega una consulta proveniente del usuario y éste identifica que fuentes están involucradas.	Recibir una consulta y que el sistema identifique que fuentes responden a esa consulta	8
H02	COMO usuario quiero que el sistema descomponga la consulta en subconsultas a las fuentes que contienen información relevante	Al sistema llega una consulta proveniente del usuario, a partir de la identificación de fuentes este descompone la consulta en subconsultas a las fuentes.	Recibir una consulta y que el sistema descomponga la consultas a subconsultas a las fuentes de datos involucradas	7
H03	COMO usuario quiero que el sistema dé respuesta a dos tipos de operadores involucrados en una consulta	Al sistema llega una consulta proveniente del usuario, éste debe identificar que operador contiene la consulta para otorgar una respuesta de acuerdo a esto.	Recibir una consulta y que el sistema identifique el tipo de operador involucrado	9

Tabla 3: Product Backlog, módulo de consultas.

El documento completo de *Product Backlog* se encuentra en el Anexo B.

3.3.1.3 Arquitectura del mediador

La arquitectura funcional del mediador propuesta se describe en la figura 14, mediante un diagrama de componentes de *UML*. En ésta se tiene un nodo de procesamiento que

incluye los componentes encargados de realizar el procesamiento de las consultas, de la integración de las respuestas, de la comunicación con las fuentes, y de la representación del modelo global en *Neo4j*. También se muestra un módulo donde se encuentran los componentes para el manejo de la base de datos relacional que contiene la representación de los mapeos a las fuentes y los mapeos de integración. En este diagrama no se muestran las fuentes ni la interfaz de usuario.

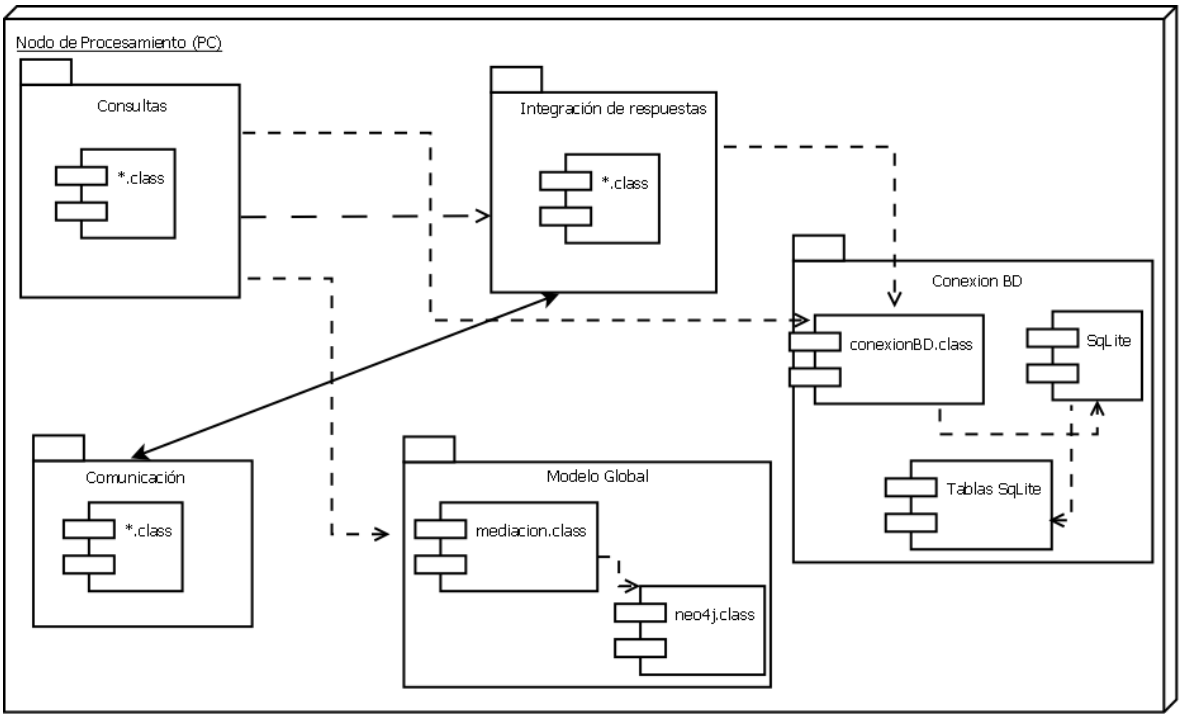


Figura 14: Descripción proceso de SCRUM, fuente [26].

3.3.1.4 Módulos del Sistema

El mediador cuenta con los módulos de consulta, comunicación, integración y respuestas.

- Módulo de Procesamiento de consultas

Este módulo es el encargado procesar las consultas provenientes del usuario. Recibe la consulta, identifica el tipo de operador y sus parámetros, identifica cuáles fuentes de datos se deben involucrar para responder la consulta, divide la consulta en subconsultas con una sintaxis estilo SPARQL que cada fuente está en capacidad de responder haciendo uso de la representación de mapeos. El procesamiento de consultas se implementó siguiendo la especificación dada en [30].

- Módulo de Comunicación

Este módulo es el encargado de establecer la comunicación con las fuentes de datos para enviar las consultas del usuario y para recibir las respuestas de la ejecución de estas

consultas. Este no hace parte del alcance del proyecto. No se implementó la conexión con las fuentes de datos para el envío de las subconsultas y para recibir las respuestas.

- Módulo de Integración de respuestas

Este módulo es el encargado de agregar los mapeos de integración descritos en la sección 3.7.2, enviar las consultas al módulo de comunicación para que estas sean enviadas a las fuentes y de integrar las respuestas de la ejecución de las consultas de las fuentes, que han sido recibidas por el módulo de comunicación. La funcionalidad de integrar los resultados de la ejecución de las consultas en las fuentes no se implementó debido a que no hace parte del alcance de este proyecto.

En la figura 15 se presenta un diagrama de paquetes en *UML* que muestra los módulos anteriormente descritos. También se incluyen las fuentes y la interfaz de usuario.

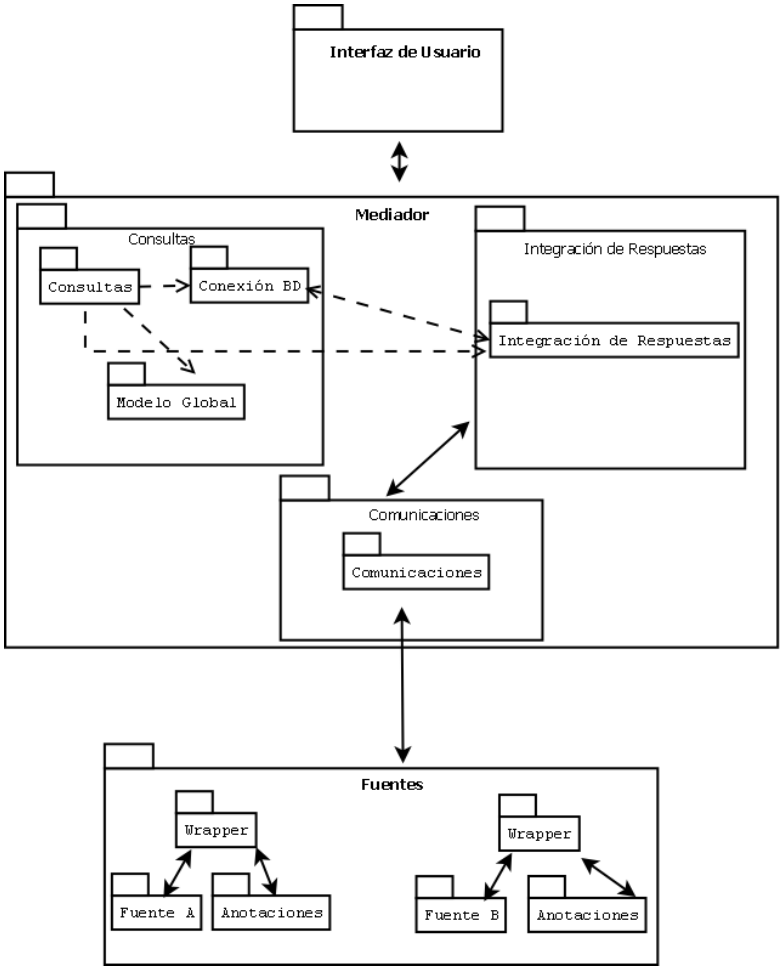


Figura 15:. Módulos del Sistema

3.3.2 Juego o Desarrollo

El propósito de esta fase en SCRUM es implementar un sistema en una serie de iteraciones conocidas como "*Sprints".* Es aquí donde el desarrollo de software se lleva a

cabo. Cada *Sprint* consta de las siguientes actividades: Elaborar, integrar, revisar y ajustar. Para el proyecto como inicio de esta fase se tiene un documento con la planeación de las iteraciones. Este documento relaciona los requisitos del *Product Backlog* a realizar en cada *Sprint.* La tabla 4 muestra el *Sprint 1* para el proyecto.

	***Sprint 1:*identificación de consulta**	**Documento: SP-001**	**Revision:001**	**Inicio:04/02/13 Fin:03/03/13**
Pila del Sprint				
ID	**Tarea**	**Prioridad**	**Duración**	**Responsable**
HC03	Definir formato de consultas, identificación del tipo de operador, parámetros de consulta, nodos de respuesta y el patrón de consulta.	9	4 semanas	Equipo de Desarrollo

Tabla 4: Sprint 1, identificación de consulta.

El documento completo de *Sprints* se encuentra en el Anexo C.

3.3.3 Pos-juego

En esta fase es el propósito que define SCRUM es el despliegue operacional. En esta fase tiene lugar las actividades de *debugging, marketing* y promoción. Al finalizar esta fase el proyecto queda terminado. Para el proyecto en esta fase no se definió ningún artefacto.

3.4 Modelo Global

La representación del modelo global se hizo con GDM (*Graph Data Model*), un modelo basado en grafos que se describe brevemente a continuación.

GDM (Graph Data Model)

GDM [27] es un modelo de base de datos de grafos basado en GOOD (Graph *Object-Oriented Data Model).* GOOD es un modelo de datos de grafos orientados a objetos. GDM consta de un grafo esquema y un grafo instancia, ambos son descritos por grafos etiquetados.

Grafo Esquema

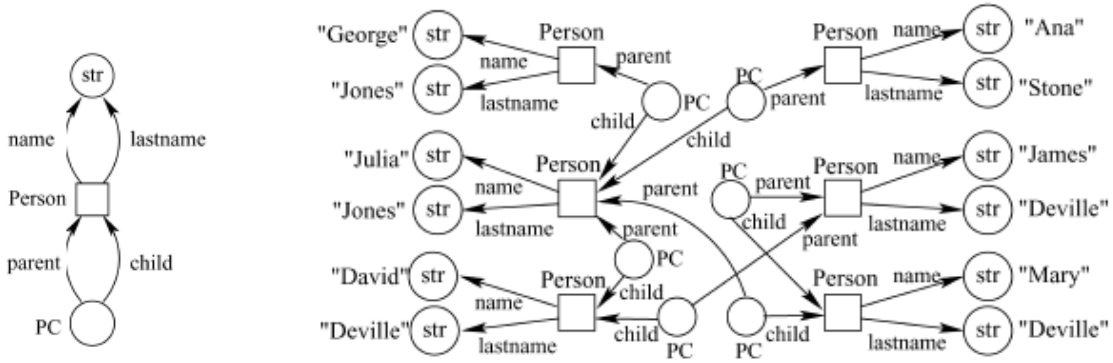
En el grafo esquema los nodos representan clases y las aristas los atributos de las clases. Los nodos se etiquetan con el nombre de la clase y las aristas con los nombres de los atributos.

Hay 3 tipos de nodos: clase objeto, clase valor compuesto y clase valor básico. Los nodos clase objeto se representan por un cuadrado, los nodos clase valor compuesto por un circulo pequeño y los nodos clase valor básico por un circulo grande con una etiqueta que indica el nombre de un tipo básico (por ejemplo str para el tipo de dato *String*).

Grafo Instancia

El grafo instancia tiene nodos que representan entidades y atributos que representan características de las entidades.

La figura 16(a), muestra un ejemplo de un esquema definido en GDM. El nodo "*Person*" es una clase objeto que tiene los atributos "*name*" y "*lastname*" (nodos clase valor básico) de tipo *String*. El nodo "*PC*" (clase valor compuesto) representa una relación, cuyos atributos son "*parent"* y "*child"*. La figura 16(b) muestra una instancia de este esquema.



(a) Grafo Esquema **(b) Grafo Instancia**

Figura 16: Modelo GDM, fuente [27]

3.5 Justificación de la implementación del modelo global en Neo4j

Para la implementación del modelo global se utilizó Neo4j. A diferencia de las otras herramientas estudiadas (Tabla 1), Neo4j reúne las características necesarias para esta implementación. Estas características son: soporte de SPARQL, ya que las consultas hacia las fuentes se expresan en ese lenguaje, código abierto, con fines académicos y soporte de algoritmos comunes sobre grafos como búsqueda de caminos entre dos nodos. Esto último se hace necesario ya que al momento de procesar una consulta que involucra diferentes nodos, se deben hallar los caminos posibles entre esos nodos. En la sección cuatro de este capítulo se describen los operadores que se implementan en este mediador.

3.6 Representación del modelo global en Neo4j

Neo4j almacena los datos en nodos conectados por relaciones. Las relaciones son dirigidas y tipadas. Tanto los nodos como las relaciones pueden tener propiedades. Estas propiedades son de la forma *key* (o nombre de la propiedad) y valor. El modelo de datos utilizado en Neo4j es conocido como *Property Graph* (Grafo atribuido)[3].

La figura 17 muestra un ejemplo de un modelo de datos en Neo4j. Los nodos tienen diferentes propiedades como *name*, *last name*, *age*, etc. Las relaciones son de tipo *KNOWS y CODED_BY* y, al igual que los nodos, poseen diferentes propiedades como *age* y *disclosure*.

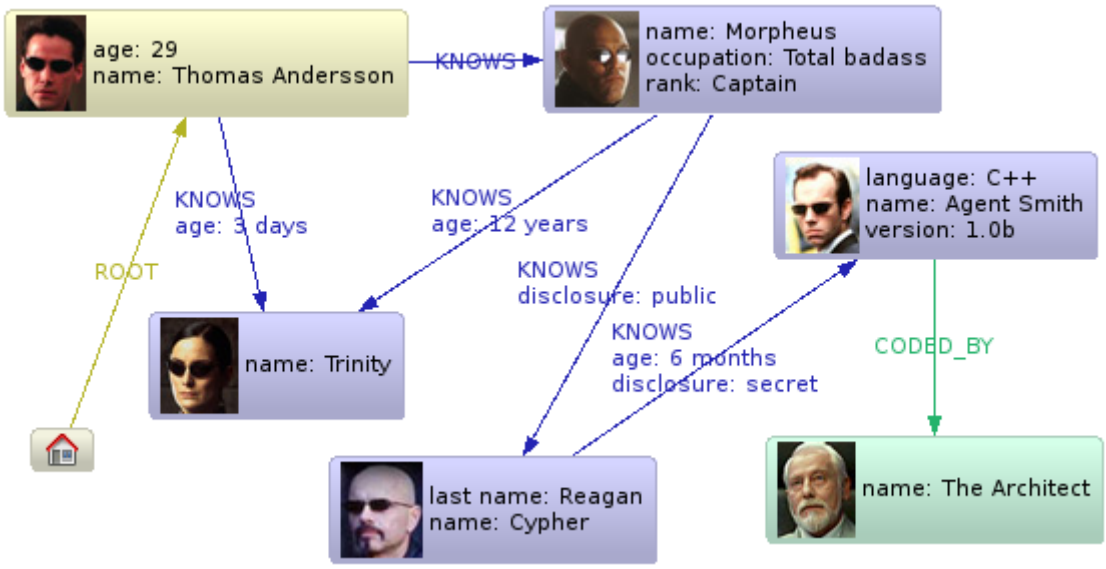


Figura 17: Modelo de datos en Neo4j[4]

Neo4j se utilizó como aplicación embebida en código java. La forma como se representaron los diferentes elementos de un modelo GDM en Neo4j fue la siguiente:

- Cada uno de los tipos de nodo de GDM se representa como nodos en Neo4j. Para cada nodo se guarda el atributo *nombre,* que hace referencia a la etiqueta del nodo en el grafo GDM, el atributo *tipoNodo,* que representa el tipo de nodo GDM que se está referenciando, ya sea nodo clase objeto, nodo clase valor compuesto

[3] http://www.neo4j.org/learn/neo4j

[4] http://blog.neo4j.org/2010/02/top-10-ways-to-get-to-know-neo4j.html

o nodo clase valor básico y *idNodo* utilizado para identificar al nodo de manera única en el modelo global. También se guarda el atributo *fuenteOrigen,* que contiene una lista de las fuente donde se almacena información de las entidades de la clase del nodo.

- Los arcos de GDM se representan en Neo4j por medio de relaciones que involucran los nodos que participan en ella. Para cada relación se guarda los nodos involucrados en ella, el atributo *etiqueta* que hace referencia a la etiqueta del arco en GDM, el atributo *tipoArco (tipo 1 o 2)* que se requiere en el proceso de generación de subconsultas descrito en el capítulo 4 y *idArco* que es utilizado para identificar de manera única a cada arco en el modelo global.

3.7 Representación de mapeos

Los mapeos se representaron siguiendo la especificación dada en [30]. En esta especificación, el modelo global del mediador es visto como una intersección de subgrafos, cada uno de los cuales representa la información de una fuente de datos. A su vez, cada fuente de datos está representada por un grafo local que se mapea directamente a la estructura de datos de la fuente. Los mapeos entre el grafo global y el grafo local de una fuente son de dos tipos: mapeos de arcos y mapeos de integración. Un arco del grafo global se mapea con un patrón de consulta (conjunto de triplas) del grafo local. Los datos provenientes de las diferentes fuentes se integran por los identificadores de las clases que las fuentes tienen en común. Los mapeos de integración definen para cada clase un patrón que permite recuperar, de la fuente, los datos de los atributos que identifican de la clase.

3.7.1 Mapeo de los Arcos del Grafo Global

Los arcos de grafo global y los grafos locales se describen como triplas (nodo, atributo, nodo). En las triplas se antepone un signo de interrogación a las etiquetas de los nodos para poder identificar las variables de la consulta.

Una expresión de mapeo de arco tiene la forma: $(n,a,m) := F[t_1\ t_2\ \ldots\ t_k]$, *donde* (n,a,m) representa un arco del modelo global*, F* una fuente de datos, y los arcos, $t_1\ t_2\ \ldots\ t_k$, definen una vista sobre el modelo local de la fuente *F*. A manera de ejemplo si se tiene *(Patient, lastName, ?paLastName)* un arco del modelo global*,* una expresión de mapeo para este arco en una fuente *F* sería: *F [?Person peLastName ?paLastName . ?Person peID ?paID . ?Patient paID ?paID],* donde la vista se describe con la sintaxis de SPARQL para definir patrones, las variables van precedidas de signo de interrogación (“?”) y cada tripla separa de otra por un punto.

La representación de mapeos se guarda en una tabla relacional, definida de la siguiente forma:

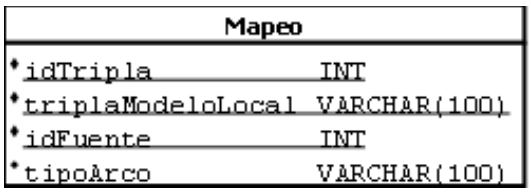


Tabla 5: tabla relacional mapeos

Donde, *idTripla* representa la identificación del arco del modelo global, *triplaModeloLocal* el arco del modelo local, *idFuente* la identificación de la fuente y *tipoArco* que representa qué tipo de arco es el que se mapea. Los tres primeros valores de esta tabla hacen parte de la llave primaria, ya que para cada arco del modelo global puede haber varios arcos en el modelo local de una fuente que definen la vista, si hay más de un arco habrá una tupla en la tabla por cada arco, es decir no se guarda toda la vista completa en el atributo *triplaModeloLocal,* se guarda arco por arco.

3.7.2 Mapeos de Integración

Cuando los subgrafos de dos fuentes se intersectan en el grafo global, los datos se deben integrar. Para ello se usan los identificadores de las clases que ambas fuentes tienen en común. La integración se realiza unificando los nodos de valor básico que identifican las entidades de cada clase. Es decir, cuando los atributos de identificación de una entidad, que proceden de fuentes diferentes, tienen el mismo valor, estos se convierten en un solo nodo, creándose así los enlaces entre las respuestas de las dos fuentes. Los atributos que identifican los objetos de cada clase se incluyen en todas las consultas, ya que son también necesarios para integrar datos de una misma fuente, cuando se genera más de una subconsulta para la fuente. Por tanto, para cada nodo de clase objeto o valor compuesto con etiqueta del grafo global se guardan los mapeos definidos por los grafos locales de las fuentes para obtener el ID del objeto.

Por ejemplo, para integrar datos de pacientes se requiere que los pacientes tengan un identificador homogéneo en todas las fuentes. Cuando las consultas hagan referencia a atributos de paciente, en las subconsultas de las fuentes se deberá incluir el identificador de paciente, con el fin de hacer integración de los datos. Si el identificador del paciente es *paID*, se definen el mapeo que permiten recuperar este atributo en cada fuente que tiene la clase paciente:

Patient: F1[?Patient paID ?paID], F2[?Patient paID ?paID]

Este mapeo indica que el atributo de identificación de *Patient* tanto en F1 como en F2 es *paID*.

Los mapeos de integración se almacenan en una tabla relacional de la siguiente forma:

Integracion	
idNodo	INT
idFuente	INT
triplaIntegradora	VARCHAR(100)

Tabla 6: tabla relacional integración

donde, el *idNodo* representa el identificador del nodo clase objeto o valor compuesto etiquetado, *idFuente* el identificador de la fuente para la que se define el mapeo de integración y *triplaIntegradora* que representa una tripla del camino que une el nodo clase objeto con su atributo identificador. Ese camino puede estar compuesto por una o más triplas Los atributos idNodo e idFuente conforman la llave primaria. Por lo que, si hay más de un tripla que conforma el mapeo integración, el camino estará conformado por todas las triplas que tengan el mismo idNodo e idFuente, es decir cada tripla del camino se guarda en un registro aparte.

Para la gestión y almacenamiento de las tablas relacionales anteriormente descritas se usó SQLite[5] como motor de base de datos

3.8 Representación de las fuentes de datos

La información de las fuentes esta almacenada en una tabla relacional, que tiene los atributos *idFuente,* identificador de la fuente, *nombre, URL* para establecer comunicación con la fuente y *descripción* donde se guarda alguna información especial de la fuente (esquema lógico, tipo de fuente). La comunicación con la fuente está por fuera del alcance de este proyecto.

Fuente	
idFuente	INT
nombre	VARCHAR(100)
url	VARCHAR(100)
descripcion	VARCHAR(500)

Tabla 7: tabla relacional fuente

3.9 Evaluación de diferentes formas de implementación para el modelo global

Como se había mencionado anteriormente para la implementación del modelo global se usa Neo4j. Dentro de las posibilidades que brinda Neo4j para representar este modelo, se evaluaron dos formas de implementación de la solución propuesta. En ellas la

[5] http://www.sqlite.org/index.html

representación de los nodos no varía, los diferentes tipos de nodo en GDM se representan como nodos objeto en Neo4j. La diferencia radica en la información de las fuentes de datos. En las secciones 3.1 y 3.2 se detallan las dos opciones de implementación.

3.9.1 Tipo de fuente como un atributo tipo cadena

Para cada nodo del modelo global es necesario tener un atributo que indique en cuál fuente se puede obtener la información correspondiente a la entidad que representa el nodo. Una de las posibilidades para guardar este atributo, era almacenarlo como un atributo de tipo *string*. Es decir, cada nodo tendrían un atributo *tipoFuente,* donde su valor podría ser “f1” o “f2”, fuente de datos relacional y fuente de datos DICOM, respectivamente para un caso particular.

Haciendo un análisis general de esta forma de implementación, se encuentra que no es flexible. A la hora de añadir una nueva fuente de datos en un futuro, habría crear una nueva comparación en el código para poder identificar esta fuente cuando se esté referenciando, además de las acciones correspondientes para poder acceder a la fuente de datos.

3.9.2 Tipo de fuente como un atributo tipo objeto

Otra de las posibilidades de implementación que se consideró fue almacenar el atributo *tipoFuente* como un objeto fuente. Este objeto fuente haría parte de un conjunto de fuentes que serían las consideradas para la información representada en el modelo global. Cada vez que se hiciera necesario añadir una nueva fuente solo era necesario modificar el conjunto de fuentes. Como se puede evidenciar esta forma de implementación es más flexible que la anterior ya que solo son necesarios pocos cambios en el código, además que al trabajar la fuente como un objeto se pueden almacenar diferentes características de cada fuente como son el tipo, un identificador, y un atributo que permita establecer la comunicación con ella.

Capítulo 4

En este capítulo se describe el funcionamiento del mediador y se presentan los resultados de las pruebas de funcionamiento. En la primera sección se describen los operadores implementados y el proceso de generación de subconsultas para cada uno. En la segunda sección se muestra un ejemplo completo de consultas para cada uno de los operadores. Finalmente, en la última sección se presentan los resultados de las diferentes pruebas realizadas.

4.1 Operadores

Para la implementación del mediador se incluyeron dos operadores: extracción de subgrafo y filtro de valor [31]. Estos operadores se implementaron siguiendo el proceso descrito en [30]. Cuando las consultas llegan al mediador, éste debe identificar el operador y los argumentos de entrada. El proceso de generación de suconsultas cambia para cada operador. Durante este proceso se distingue en el modelo global dos tipos de arco, los que conectan una clase objeto o de valor compuesto etiquetada con una clase de valor básico (arcos tipo 1), y las demás combinaciones de tipos de nodo posibles (arco tipo 2).

Las subconsultas que genera el mediador se expresan con una sintaxis similar a la del *Select Query* de SPARQL. Sin embargo, el *SELECT* incluye solamente la lista de datos a retornar y el *WHERE* incluye un grupo de patrones de arcos que, además de los arcos, pueden incluir solamente las cláusulas *FILTER* y *OPTIONAL*. La siguiente es la estructura de una subconsulta que incluye la cláusula *FILTER:*

SELECT datos _a_retornar *WHERE* patrón_consulta *FILTER* expresiones_de_filtro

A continuación se describen los operadores implementados y la forma como se generan las subconsultas para cada uno.

4.1.1 Operador de Extracción de Subgrafos

El operador de extracción de subgrafos tiene como entrada un conjunto de nodos de interés del esquema (modelo Global). Este operador busca en el esquema los caminos no dirigidos entre los nodos de interés para construir con ellos el esquema de respuesta. La instancia de respuesta incluye todos los nodos que pertenecen a las clases del esquema respuesta y los arcos entre ellos. Este operador se expresa de la siguiente forma:

Extract({m1, m2, …, mk})

donde *{m1, m2, … , mk}* son nodos del grafo esquema de interés para el subgrafo.
La salida de este operador es el subgrafo que incluye los nodos y arcos de todos los caminos entre los nodos de entrada *{m1, m2, … , mk}.*

4.1.1.1 Generación de subconsultas para el operador de extracción de subgrafos

Para calcular la salida del operador **extracción de subgrafos** se llevan a cabo los siguientes pasos:

1. Se calculan los caminos de los nodos *{m1, m2, …, mk}* en el esquema. La unión de estos caminos constituye el subgrafo del esquema. Para el cálculo de estos caminos se usó el paquete "*graph-algo*", un componente de Neo4j que implementa algunos algoritmos frecuentemente usados en grafos.
2. A partir del subgrafo esquema calculado en el paso 1, para cada fuente contenida en la tabla relacional fuente (Tabla 5), se identifican los arcos que se mapean en ella y las clases objeto o de valor compuesto etiquetadas del modelo global que estos arcos incluyen.
3. Para cada fuente se generan consultas por cada clase objeto o valor compuesto etiquetada (arcos tipo 1) y para cada relación entre clases objeto y clases valor compuesto etiquetadas (arcos de tipo 2). El patrón de consulta de las clases objeto o valor compuesto etiquetado se genera concatenando el mapeo que permite obtener el atributo identificador del objeto de esa clase en la fuente(mapeos de integración), con los mapeos definidos para cada arco de tipo 1 que incluyen esa agregados con la cláusula "OPTIONAL" de SPARQL. El patrón de consulta para la relación entre clases objeto y valor compuesto etiquetadas se genera concatenando los mapeos que permiten obtener los atributos identificadores de los objetos de las clases relacionadas (mapeos de integración) y el mapeo definido para ese arco.

A partir de cada patrón de consulta construido se construye una subconsulta para la fuente de la siguiente forma:

- La lista del **SELECT** incluye todas las variables que hacen referencia a nodos de clase valor básico
- En el **WHERE** aparece el patrón de consulta que se construyó

En este operador no hay filtro de consulta.

4.1.2 Operador filtro de valor

Este tipo de operador tiene como entrada un conjunto de nodos de interés del esquema y una expresión de la forma *mb op v* (*m* es un nodo del esquema, *op* es un operador de comparación y *v* un valor). Cada nodo de interés tienen asociado un camino que lo une con el nodo de la expresión. Este operador permite seleccionar datos de valor básico que están conectados por un camino con un nodo de valor básico que cumple con la condición especificada en la expresión. Este operador se expresa de la siguiente forma:

ValueFilter ({m1{p1}, m2{p2}, … mk{pk} }, exp)

donde *{m1, m2, … , mk}* son nodos del grafo esquema de interés para la respuesta. *{{p1},{p2},…,{pk}}* son los caminos que unen los nodos de interés con el nodo *mb* de la condición de la expresión *exp*.

4.1.2.1 Generación de subconsultas para el operador filtro de valor

Este operador genera una subconsulta por cada camino *{{p1},{p2},…,{pk}}* para cada fuente relevante al camino (fuente que aporta información de los nodos y arcos del camino), por lo que cada fuente puede recibir varias subconsultas. Los pasos seguir para la generación de las subconsultas de cada camino son los siguientes:

1. Para cada fuente contenida en la tabla relacional fuente (Tabla 5) se identifican los arcos que se mapean en ella y las clases objeto o de valor compuesto etiquetadas (del modelo global) que estos arcos incluyen.
2. A partir del paso anterior, para cada fuente se genera una consulta por cada clase objeto o valor compuesto etiquetada (arcos tipo 1), y una consulta para cada relación entre clases objeto y clases valor compuesto etiquetadas (arcos tipo 2). El patrón de arcos de las clases objeto o valor compuesto etiquetadas se genera concatenando el mapeo que permite obtener el atributo identificador del objeto de esa clase en la fuente(mapeos de integración) con los mapeos definidos para cada arco de tipo 1 que incluyen esa clase. En este caso no se agregan con la cláusula OPTIONAL. El patrón de consulta para la relación entre clases objeto y valor compuesto etiquetadas se genera concatenando los mapeos que permiten obtener los atributos identificadores de los objetos de las clases relacionadas en la fuente (mapeos de integración) y el mapeo definido para ese arco. El *FILTER* con la expresión *exp* del *valueFilter* se agrega en los patrones de consulta de aquellos arcos que incluyen variables que se usan en el filtro de la consulta. *exp* se debe adaptar para anteponer el signo de interrogación al nombre de la variable.

A partir de cada patrón de consulta construido se forma una subconsulta para la fuente de la siguiente forma:

- La lista en el **SELECT** incluye todas las variables que se refieren a nodos de clase valor básico
- En el **WHERE** aparece el patrón de consulta que se construyó
- El **FILTER** con la expresión *exp* del *valueFilter*.

4.2 Ejemplos de consultas

A continuación se presenta una consulta para cada tipo de operador implementado [30]. Para este ejemplo se cuenta con una definición del modelo global en GDM sobre el cual se plantean las consultas, los modelos locales de las fuentes en GDM, que no se almacenan en ningún componente de la arquitectura. Estos son solo una representación de los datos que contiene cada fuente y la forma como se almacenan. A partir del modelo global y los grafos locales se obtiene la representación de mapeos que también se detalla.

4.2.1 Modelo Global GDM

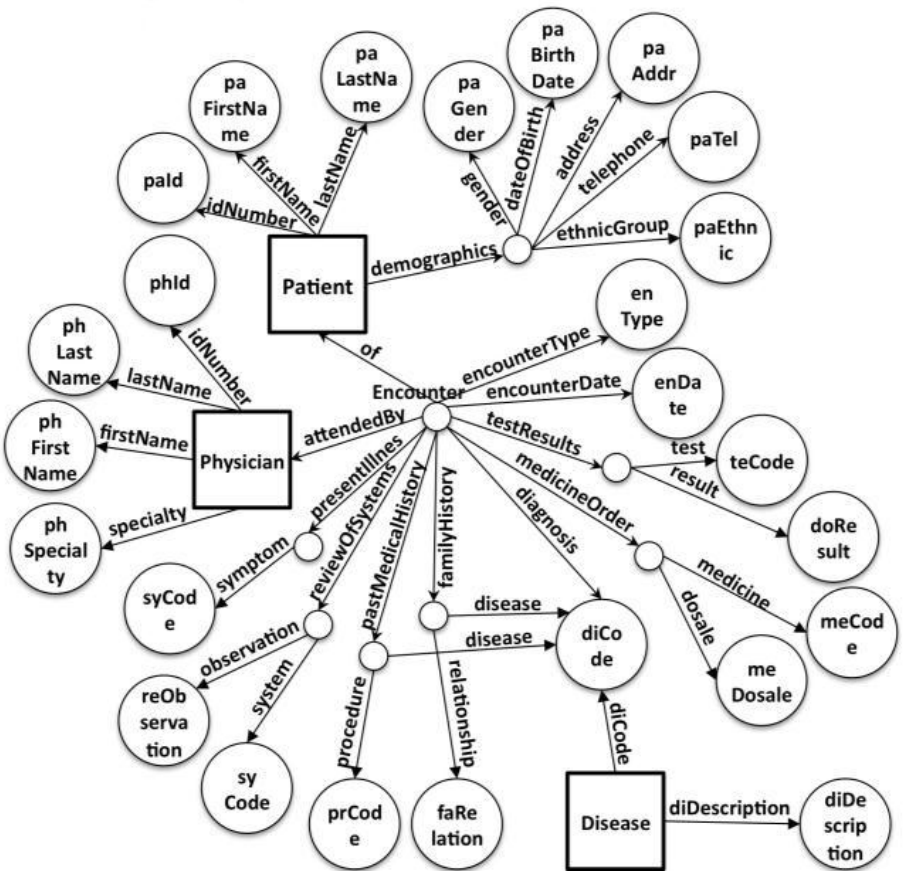


Figura 18: Esquema Modelo Global, fuente [30].

4.2.2 Grafos Locales

En este ejemplo se tienen tres fuentes S1, S2 y S3. Dos o más fuentes pueden almacenar información común. La relación entre la información que almacena cada fuente y el modelo global se muestra en la figura 19. Los grafos locales de las fuentes S1, S2, y S3 se muestran en las figuras 20,21 y 22, respectivamente.

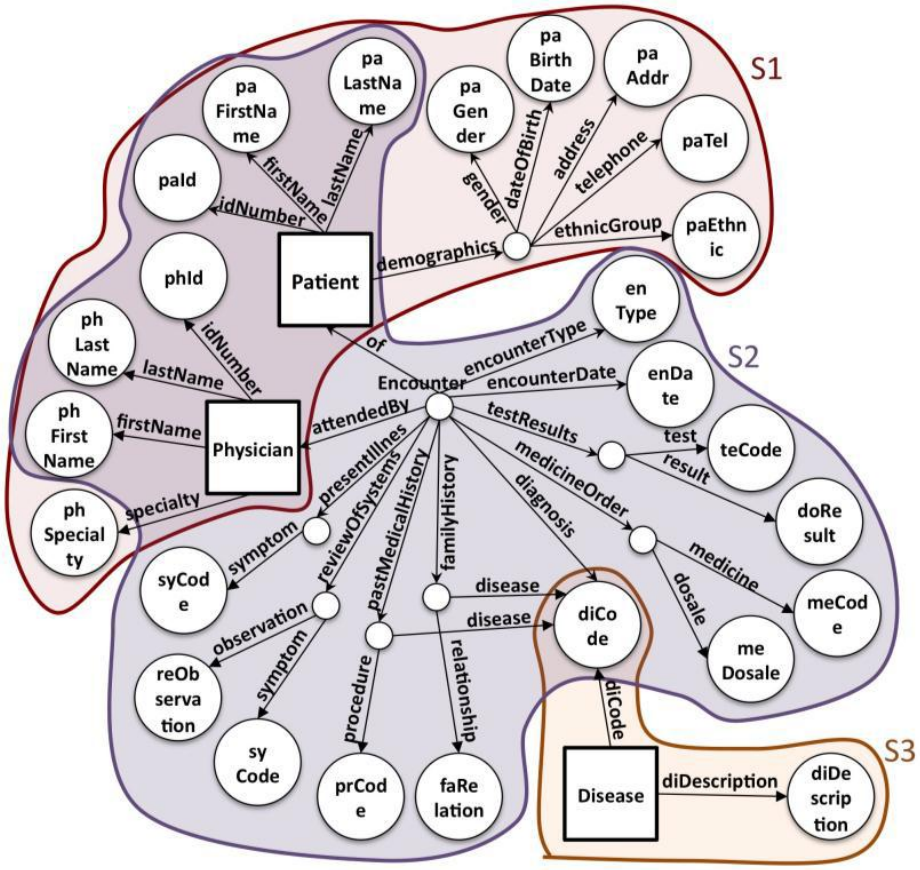


Figura 19: Relación entre el modelo global y las fuentes, fuente [30].

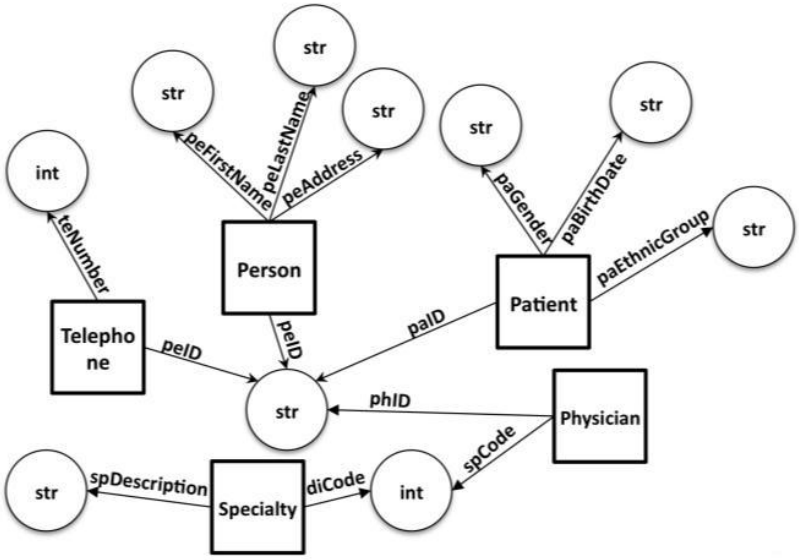


Figura 20: Grafo local fuente S1, fuente [30].

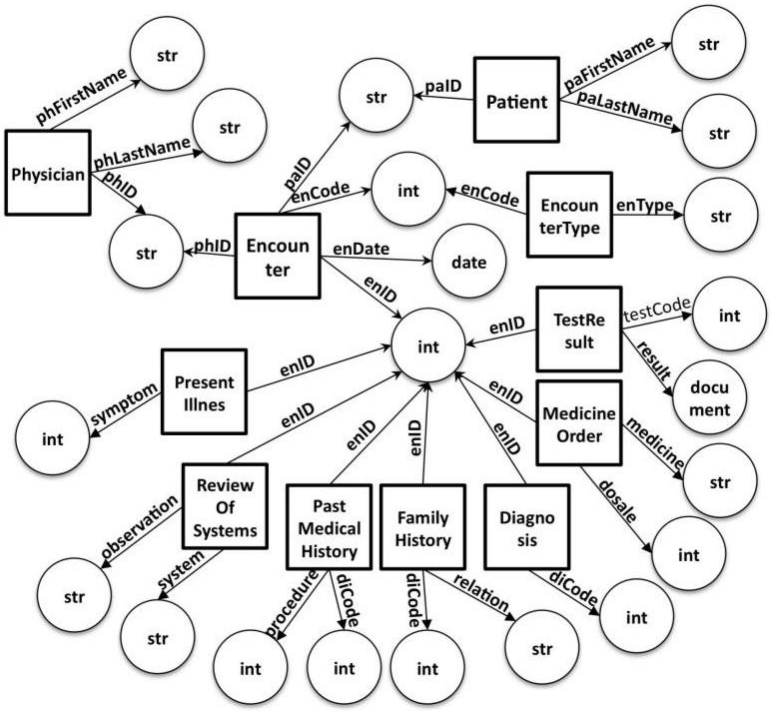


Figura 21: Grafo local fuente S2, fuente [30]

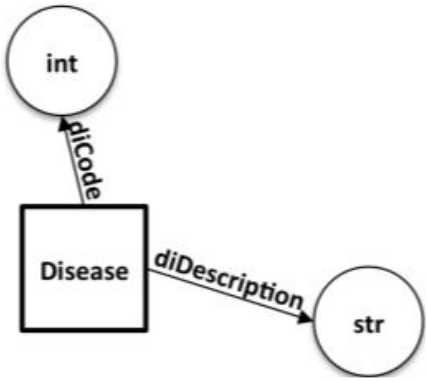


Figura 22: Grafo local fuente S3, fuente [30].

4.2.3 Representación de mapeos

Con base en el esquema del modelo global y los esquemas locales de las fuentes se representan los mapeos. A continuación se presenta, de forma general, esta representación.

4.2.2.1 Mapeos de integración

Los nodos clase objeto y clase valor compuesto con etiqueta (relaciones) del modelo global se mapean en los grafos locales para obtener el ID del objeto (Mapeos de Integración).

- Patient: S1[?Patient paID ?paID], S2[?Patient paID ?paID]
- Physician: S1[?Physician phID ?phID], S2[?Physician phID ?phID]
- Disease: S3[?Disease diCode ?diCode]
- Encounter: S2[?Encounter enID ?enID]

4.2.2.2 Mapeos arcos de tipo 1

Son los arcos que unen una clase objeto o de valor compuesto etiquetada con una clase de valor básico.

- (Patient, lastName, ?paLastName) := S1[?Person peLastName ?paLastName . ?Person peID ?paID]
- (Patient, lastName, ?paLastName) := S2[?Patient paLastName ?paLastName]
- (Encounter, encounterDate, ?enDate) := S2[?Encounter enDate ?enDate]
- (Disease, description, ?diDescription) := S3[?Disease diDescription ? diDescription]
- (Physician, lastName, ?phLastName) := S1[?Person peLastName ?phLastName . ?Person peID ?phID]
- (Physician, lastName, ?phLastName) := S2[?Physician phLastName ?phLastName]
- (Physician, specialty, ?phSpecialty) := S1[?Physician spCode ?spCode . ?Specialty spCode ?spCode . ?Specialty spDescription ?phSpecialty]

4.2.2.3 Mapeos arcos de tipo 2

Son los arcos que unen dos clase objeto o de valor compuesto etiquetada.

- (Encounter, of, Patient) := S2[?Encounter paID ?paID]
- (Encounter, attendedBy, Physician) := S2[?Encounter phID ?phID]

Los arcos que incluyen clases de valor compuesto no etiquetadas

- (Patient, demographics, ß). ß representa el nodo sin etiqueta. Este arco no tendría mapeo definido. El mapeo estaría definido por ejemplo para el arco (ß, ethnicGroup, ?paEthnic) := S1[?Patient paEthnicGroup ?paEthnic] .Donde ß en este caso hace referencia a la clase Patient.

4.2.3 Ejemplo consulta operador de extracción de subgrafos

- **Consulta:** Extract ({paLastName, phLastName, diDescription})

1. Caminos entre los nodos de interés y subgrafo del esquema

- **De paLastName a diDescription**

p1{(Patient,lastName,paLastName),(Encounter,of,Patient),(Encounter,diagnosis,diCode),(Disease,diCode,diCode), (Disease,description,diDescription)}
p2{(Patient,lastName,paLastName),(Encounter,of,Patient),(Encounter,familyHistory,ß1),(ß1,disease,diCode)(Disease,diCode,diCode),(Disease,description,diDescription)}
p3{(Patient,lastName,paLastName),(Encounter,of,Patient),(Encounter,pastMedicalHistory,ß2),(ß2,disease,diCode)(Disease,diCode,diCode),(Disease,description,diDescription)}

- **De phLastName a diDescription**

p4{(Physician,lastName,phLastName),(Encounter,attendedBy,Physician),(Encounter,diagnosis,diCode),(Disease,diCode,diCode),
(Disease,description,diDescription)}
p5{(Physician,lastName,phLastName),(Encounter,attendedBy,Physician),(Encounter,familyHistory,ß1),(ß1,disease,diCode),
(Disease,diCode,diCode),(Disease,description,diDescription)}
p6{(Physician,lastName,phLastName),(Encounter,attendedBy,Physician),(Encounter,pastMedicalHistory,ß2),(ß2,disease,diCode),
Disease,diCode,diCode),(Disease,description,diDescription)}

- **De paLastName a phLastName**
p7{(Patient,lastName,paLastName),(Encounter,of,Patient),(
Encounter,attendedBy,Physician),(Physician,lastName,phLastName)}

- **Subgrafo del esquema que resulta de la unión sin duplicados de todos los caminos**

{(Patient,lastName,paLastName),(Encounter,of,Patient),(Encounter,diagnosis,diCode),(Disease,diCode,diCode),(Disease,description,diDescription),(Encounter,familyHistory,ß1),(ß1,disease,diCode),(Encounter,pastMedicalHistory,ß2), (ß2,disease,diCode),(Physician,lastName,phLastName),(
Encounter,attendedBy,Physician)}

2. Mapeos en cada fuente e identificación de las clase objeto y de valor compuesto etiquetadas

	Subgrafo Global	S1	S2	S3

1	(**Patient**, lastName, ?paLastName)	?Person peLastName ?paLastName . ?Person peID ?paID	?Patient paLastName ?paLastName	
2	(**Encounter**, of, **Patient**)		?Encounter paID ?paID	
3	(**Encounter**, diagnosis, ?diCode)		?Diagnosis enID ?enID . ?Diagnosis diCode ?diCodeD	
4	(**Disease**, diCode, ?diCode)			
5	(**Disease,** description, ?diDescription)			?Disease diDescription ?diDescription
6	(Encounter, familyHistory, $ß_1$)			
7	($ß_1$, disease, ?diCode) (el $ß_1$ agrega la clase **Encounter**)		?FamilyHistory enID ?enID . ?FamilyHistory diCode ?diCodeFh	
8	(Encounter, pastMedicalHistory, ß2)			
9	($ß_2$, disease, ?diCode) (el $ß_2$ agrega la clase **Encounter**)		?PastMedicalHistory enID ?enID . ?PastMedicalHistory diCode ?diCodeMh	
10	(**Physician**,lastName,?phLastName)	?Person peLastName ?phLastName . ?Person peID ?phID	?Physician phLastName ?phLastName .	
11	(**Encounter**,attendedBy,**Physician**)		?Encounter phID ?phID	

3. Patrones de las subconsultas y subconsultas a las fuentes

Las clases objeto y valor compuesto etiquetadas que cada fuente incluye:

Para S1: Incluye la clase Patient (arco de la fila 1) y Physician (arco de la fila 10). Se generan dos consultas una para Patient y una para Physician. Ambos arcos son de tipo 1, por lo que lo el patrón de consulta es el siguiente:

1. { ?Patient paID ?paID . OPTIONAL { ?Person peLastName ?paLastName . ?Person peID ?paID } }
2. { ?Physician phID ?phID . OPTIONAL { ?Person peLastName ?phLastName . ?Person peID ?phID } }

Para S2: Incluye la clase Patient (arcos de la fila 1 y 2), Encounter (arcos de la fila 2, 3, 7,9 y 11) y Physician (arcos de la fila 10 y 11). Los arcos 1, 3, 7, 9, 10 son tipo 1. Los arcos 2 y 11 son tipo 2, representan la relación Patient-Encounter (2) y la relación Encounter-Physician (11).
En los arcos 7 y 9 hay nodos de valor compuesto no etiquetados. Estos traen la clase objeto o de valor compuesto etiquetado a la que pertenecen, en este caso la clase Encounter.
Se generan 5 consultas, las tres primeras inician con los mapeos de los atributos de identificación de Patient, Encounter y Physician respectivamente. Las dos últimas incluyen el mapeo de los atributos de identificación de las clases relacionadas y el mapeo del arco. Los patrones de estas consultas son los siguientes:

3. { ?Patient paID ?paID . OPTIONAL { ?Patient paLastName ?paLastName } }
4. { ?Encounter enID ?enID . OPTIONAL { ?Diagnosis enID ?enID . ?Diagnosis diCode ?diCodeD } . OPTIONAL { ?FamilyHistory enID ?enID . ?FamilyHistory diCode ?diCodeFh } . OPTIONAL { ?PastMedicalHistory enID ?enID . ?PastMedicalHistory diCode ?diCodeMh } }
5. { ?Physician phID ?phID . OPTIONAL { ?Physician phLastName ?phLastName } }
6. { ?Patient paID ?paID . ?Encounter enID ?enID . ?Encounter paID ?paID }
7. { ?Physician phID ?phID . ?Encounter enID ?enID . ?Encounter phID ?phID }

Para S3: Incluye la clase Disease (arco de la fila 5). Este arco es de tipo 1. Se genera una consulta que inicia con el mapeo identificador para la clase. El patron de esta consulta seria el siguiente:

8. {?Disease diCode ?diCode . OPTIONAL { ?Disease diDescription ?diDescription } }

Subconsultas a las fuentes

Para S1:

1. SELECT ?paID ?paLastName

WHERE { ?Patient paID ?paID . OPTIONAL { ?Person peLastName ?paLastName . ?Person peID ?paID } }
2. SELECT ?phID ?phLastName
WHERE { ?Physician phID ?phID .OPTIONAL { ?Person peLastName ?phLastName . ?Person peID ?phID } }

Para S2:

3. SELECT ?paID ?paLastName
WHERE { ?Patient paID ?paID . OPTIONAL { ?Patient paLastName ?paLastName} }
4. SELECT ?enID ?diCodeD ?diCodeFh ?diCodeMh
WHERE {?Encounter enID ?enID . OPTIONAL { ?Diagnosis enID ?enID . ?Diagnosis diCode ?diCodeD }.OPTIONAL { ?FamilyHistory enID ?enID . ?FamilyHistory diCode ?diCodeFh } .OPTIONAL { ?PastMedicalHistory enID ?enID . ?PastMedicalHistory diCode ?diCodeMh } }
5. SELECT ?phID ?phLastName
WHERE {?Physician phID ?phID . OPTIONAL { ?Physician phLastName ?phLastName } }
6. SELECT ?paID ?enID
WHERE { ?Patient paID ?paID . ?Encounter enID ?enID . ?Encounter paID ?paID}
7. SELECT ?phID ?enID
WHERE { ?Physician phID ?phID . ?Encounter enID ?enID . ?Encounter phID ?phID}

Para S3:

8. SELECT ?diCode ?diDescription
WHERE { ?Disease diCode ?diCode . OPTIONAL { ?Disease diDescription ?diDescription } }

4.2.4 Ejemplo consulta operador filtro de valor

Consulta:

ValueFilter ({paLastName$_{\{p1\}}$, enDate$_{\{p2\}}$}, diDescription = “Diabetes")
Donde:
{p1}={(Patient,lastName,paLastName),(Encounter,of,Patient),(Encounter,diagnosis,diCode),(Disease,diCode,diCode) (Disease,description,diDescription)},
{p2}={(Encounter,encounterDate,enDate),(Encounter,diagnosis,diCode),(Disease,diCode,diCode),(Disease,description,diDescription)}

1. Mapeos en cada Fuente e identificación de las clase objeto y de valor compuesto etiquetadas

	p1	S1	S2	S3

1	(**Patient**, lastName, ?paLastName)	?Person peLastName ?paLastName . ?Person peID ?paID	?Patient paLastName ?paLastName	
2	(**Encounte**r, of, **Patient**)		?Encounter paID ?paID	
3	(**Encounter**, diagnosis, ?diCode)		?Diagnosis enID ?enID . ?Diagnosis diCode ?diCodeD	
4	(**Disease**, diCode, ?diCode)			
5	(**Disease**, description, ?diDescription)			?Disease diDescription ?diDescription

	p2	**S1**	**S2**	**S3**
6	(**Encounter**, encounterDate, ?enDate)		?Encounter enDate ?enDate	
7	(**Encounter**, diagnosis, ?diCode)		?Diagnosis enID ?enID . ?Diagnosis diCode ?diCodeD	
8	(**Disease**, diCode, ?diCode)			
9	(**Disease**, description, ?diDescription)			¿Disease diDescription ?diDescription

2. Patrones de las subconsultas, aplicación del filtro y subconsultas a las fuentes

Para S1:

Camino p1

Incluye la clase Patient (arco de la fila 1). Este arco es tipo 1, por lo que el patrón de consulta seria el siguiente:

1. { ?Patient paID ?paID . ?Person peLastName ?paLastName . ?Person peID ?paID }

Camino p2
La fuente S1 no tiene mapeos en este camino.

Para S2:

Camino p1

Incluye las clases Patient (arcos de la fila 1 y 2) y Encounter (arcos de la fila 2 y 3). Los arcos 1 y 3 son tipo 1. El arco 2 es tipo 2. Se generan 3 consultas: Patient (con 1), Encounter (con 3), y la relación Patient-Encounter (con 2). Los patrones de estas consultas son los siguientes:

2. { ?Patient paID ?paID . ?Patient paLastName ?paLastName }
3. { ?Encounter enID ?enID . ?Diagnosis enID ?enID . ?Diagnosis diCode ?diCodeD }
4. { ?Patient paID ?paID . ?Encounter enID ?enID . ?Encounter paID ?paID }

Camino p2

Incluye la clase Encounter (arcos de la fila 6 y 7). Estos son arcos tipo 1. Se genera una consulta que inicia con el mapeo definido para la identificación de Encounter y se agregan los mapeos de 6 y 7. El patrón de consulta es el siguiente:

5. {?Encounter enID ?enID . ?Encounter enDate ?enDate . ?Diagnosis enID ?enID . ?Diagnosis diCode ?diCodeD }

Para S3:

Camino p1

Incluye la clase objeto Disease (arco de la fila 5). Este arco es tipo 1. Se genera una consulta que inicia con el mapeo definido para Disease. El patrón de consulta es el siguiente:

6. { ?Disease diCode ?diCode . ?Disease diDescription ?diDescription }

Camino p2

Incluye la clase objeto Disease (arco de la fila 9). Este arco es tipo 1. Se genera una consulta que inicia con el mapeo definido para Disease. El patrón de consulta es el siguiente:

7. { ?Disease diCode ?diCode . ?Disease diDescription ?diDescription }

Aplicación del filtro

El filtro se aplica sobre la clase valor básico diDescription que está en las filas 5 y 9, que se mapean a la fuente 3. Entonces la condición se aplica solamente en las subconsultas de la fuente 3 que incluyen los mapeos de 5 y 9. El filtro es de la siguiente forma:

FILTER (?diDescription = “Diabetes")

Subconsultas a las fuentes

Para S1:

1. SELECT ?paID ?paLastName
WHERE { ?Patient paID ?paID . ?Person peLastName ?paLastName . ?Person peID ?paID }

Para S2:

2. SELECT ?paID ?paLastName
WHERE { ?Patient paID ?paID . ?Patient paLastName ?paLastName }
3.SELECT ?enID ?diCodeD
WHERE { ?Encounter enID ?enID . ?Diagnosis enID ?enID . ?Diagnosis diCode ?diCodeD }
4. SELECT ?paId ?enID
WHERE { ?Patient paID ?paID . ?Encounter enID ?enID . ?Encounter paID ?paID }
5. SELECT ?enDate ?enID ?diCodeD
WHERE { ?Encounter enID ?enID . ?Encounter enDate ?enDate . ?Diagnosis enID ?enID . ?Diagnosis diCode ?diCodeD }

Para S3:

6. SELECT ?diDescription ?diCode
WHERE { ?Disease diCode ?diCode . ?Disease diDescription ?diDescription . FILTER (?diDescription = “Diabetes") }

7. SELECT ?diDescription ?diCode
WHERE { ?Disease diCode ?diCode . ?Disease diDescription ?diDescription . FILTER (?diDescription = “Diabetes") }

4.3 Pruebas y Resultados

En esta sección se describen las pruebas realizadas a la solución propuesta y se detallan los resultados obtenidos. El objetivo es comprobar el buen funcionamiento del sistema implementado para este proyecto. Para las pruebas del sistema se definió un formato de entrada para las consultas en XML y un formato de salida para las subconsultas

igualmente. A continuación se muestra la estructura de estos formatos, los casos de pruebas realizados y los resultados obtenidos.

4.3.1 Formato de entrada de consultas

Cuando la consulta involucra el operador filtro de valor el formato definido es:

```
<?xml version="1.0" encoding="utf-8"?>
<consulta>
  <operador>VALUEFILTER</operador>
  <nodosRespuesta>
    <nodo>
      <id></id>
    </nodo>
    <pk>
      <arco>
        <id></id>
      </arco>
      <arco>
        <id></id>
      </arco>
    </pk>
    <nodo>
      <id></id>
    </nodo>
    <pk>
      <arco>
        <id></id>
      </arco>
      <arco>
        <id></id>
      </arco>
    </pk>
  </nodosRespuesta>
  <filtro>
    <condicion>
      <nodo></nodo>
      <operador></operador>
      <valor></valor>
    </condicion>
  </filtro>
</consulta>
```

El nodo raíz es "consulta". Tiene un nodo "nodosRespuesta", que contiene las variables a retornar. Cada variable a retornar se referencia por su identificador en el modelo global. Por cada variable a retornar se tiene un nodo "pk" que representa el camino entre la variable y el nodo de la expresión de filtro. Los caminos tienen nodos "arco". Los arcos se referencian por su identificador en el modelo global. Por último, se tiene un nodo "filtro" conformado por el nodo "condicion" que contiene el nodo al cual se aplica el filtro, el nodo "operador" y el nodo "valor".

El XML esquema [6] definido para el anterior XML es el siguiente:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="consulta">
    <xs:complexType>
      <xs:sequence>
        <xs:element type="xs:string" name="operador"/>
        <xs:element name="nodosRespuesta">
          <xs:complexType>
            <xs:choice maxOccurs="n" minOccurs="1">
              <xs:element name="nodo">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element type="xs:string" name="id"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="pk">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="arco" maxOccurs="n" minOccurs="1">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element type="xs:string" name="id"/>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element type="xs:string" name="id" minOccurs="0"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:choice>
          </xs:complexType>
        </xs:element>
        <xs:element name="filtro">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="condicion">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element type="xs:string" name="nodo"/>
                    <xs:element type="xs:string" name="operador"/>
                    <xs:element type="xs:string" name="valor"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Cuando la consulta involucra el operador extracción de subgrafos el formato definido es:

[6] http://www.w3.org/XML/Schema

```
<?xml version="1.0" encoding="utf-8"?>
<consulta>
  <operador>EXTRACT</operador>
  <nodosRespuesta>
    <nodo>
      <id></id>
    </nodo>
    <nodo>
      <id></id>
    </nodo>
    <nodo>
      <id></id>
    </nodo>
  </nodosRespuesta>
</consulta>
```

Al igual que en el formato anterior el nodo raíz es “consulta. Tiene un nodo “nodosRespuesta”, que contiene las variables a retornar. Cada variable a retornar se referencia por su identificador en el modelo global.

El XML esquema definido para el anterior XML es el siguiente:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="consulta"><xs:complexType>
      <xs:sequence>
        <xs:element type="xs:string" name="operador"/>
        <xs:element name="nodosRespuesta">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="nodo" maxOccurs="n" minOccurs="1">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element type="xs:string" name="id"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

4.3.2 Formato de salida de subconsultas

Los dos operadores implementados generan el mismo formato de salida. Cuando se recibe y se procesa una consulta se obtiene como resultado un archivo con las consultas correspondientes a cada fuente. El formato de salida es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<consultas>
  <consulta>
    <select>
      <nodoRetorno>
        <etiqueta></etiqueta>
      </nodoRetorno>
    </select>
    <where>
      <patron>
        <arco>
          <clase></clase>
          <relacion></relacion>
          <objeto></objeto>
        </arco>
        <arco>
          <clase></clase>
          <relacion></relacion>
          <objeto></objeto>
        </arco>
        <optional>
          <arco>
            <clase></clase>
            <relacion></relacion>
            <objeto></objeto>
           </arco>
        </optional>
        <optional>
          <arco>
            <clase></clase>
            <relacion></relacion>
            <objeto></objeto>
          </arco>
        </optional>
    </patron>
    <filtro>
      <expresion>
        <dato></dato>
        <operadorComparacion></operadorComparacion>
        <valor></valor>
      </expresion>    </filtro>
  </where>  </consulta>
</consultas>
```

El nodo raíz es “consultas”. Este puede tener uno o varios nodos “consulta”. Un nodo “select” contiene las variables a retornar en nodos “nodoRetorno”. Los nodos “nodoRetorno” tendrán un nodo “etiqueta” que contiene la etiqueta del nodo valor básico a retornar. En el nodo “where” está definido el patrón de consulta, que puede incluir uno o varios nodos “arco” divididos en sus tres elementos (clase, relación y objeto), nodos “optional” o un nodo “filtro”. Si la subconsulta es resultado del procesamiento del operador extracción de subgrafos y esta requiere la inclusión de la cláusula *OPTIONAL*, se tendrán dentro del nodo “patron” uno o varios nodos “optional” y estos a su vez contendrán uno a varios nodos “arco”. Si la subconsulta es resultado del procesamiento del operador filtro

de valor, dentro del nodo “patron” no se tendrán nodos “optional”. Si esta requiere la inclusión de la cláusula *FILTER* dentro del nodo “patron” habrá un nodo “filtro.”

El XML esquema para el anterior archivo XML es el siguiente:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="consultas">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="consulta">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="select">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="nodoRetorno">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element type="xs:string" name="etiqueta"/>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="where">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="patron">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="arco" maxOccurs="n" minOccurs="1">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element type="xs:string" name="clase"/>
                                <xs:element type="xs:string" name="relacion"/>
                                <xs:element type="xs:string" name="objeto"/>
                              </xs:sequence>
                            </xs:complexType>
                          </xs:element>
                          <xs:element name="optional" maxOccurs="n"
minOccurs="0">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element name="arco">
                                  <xs:complexType>
                                    <xs:sequence>
                                      <xs:element type="xs:string" name="clase"/>
                                      <xs:element type="xs:string"
name="relacion"/>
                                      <xs:element type="xs:string"
name="objeto"/>
                                    </xs:sequence>
                                  </xs:complexType>
                                </xs:element>
                              </xs:sequence>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
```

```
                    <xs:element name="filtro">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="expresion">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element type="xs:string" name="dato"/>
                                <xs:element type="xs:string"
name="operadorComparacion"/>
                                <xs:element type="xs:string" name="valor"/>
                              </xs:sequence>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

4.3.3 Casos de prueba

En la siguiente tabla se muestran cada uno de las pruebas realizadas:

Componente	Casos	Descripción	Sección
Lectura del archivo de entrada que contiene la consulta	Extensión de archivo no valida	El sistema debe mostrar un mensaje indicando que el archivo debe ser de tipo XML	4.3.3.1
	Falta definición de algún elemento en el archivo de entrada	El sistema debe mostrar un mensaje indicando que falta algún elemento en el archivo de entrada	4.3.3.2
	Elemento del archivo de entrada no tiene valor		4.3.3.3

Procesamiento de consultas	Consulta que involucra el operador extracción de subgrafos	El sistema debe arrojar cada una de las consultas resultantes de acuerdo al proceso para el operador. Estas subconsultas debe ser expuestas en el formato de salida especificado	4.3.3.4
	Consulta que involucra el operador filtro de valor	El sistema debe arrojar cada una de las consultas resultantes de acuerdo al proceso para el operador. Estas subconsultas debe ser expuestas en el formato de salida especificado	4.3.3.5

Tabla 8: Casos de Prueba de la aplicación

4.3.3.1 Extensión de archivo no válida

La extensión válida de los mensajes de entrada en los que se expresan las consultas debe ser XML. Para probar esta funcionalidad se realizó la prueba con un archivo que no contiene esta extensión.

Cuando se ejecuta la aplicación se elige un archivo con extensión no válida (ver figura 23). En este caso un archivo con extensión “txt”. El sistema mostrará un mensaje de error notificando del tipo de archivo no válido (ver figura 24).

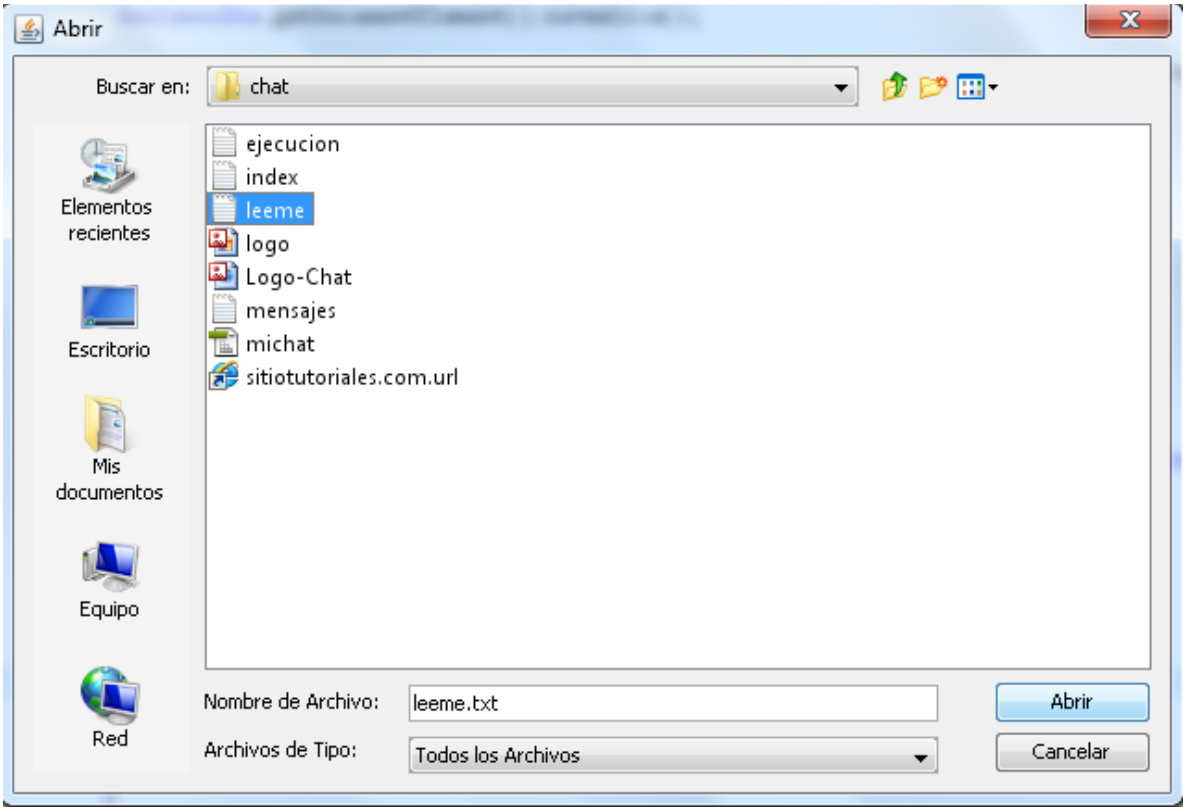


Figura 23: Selección archivo no válido

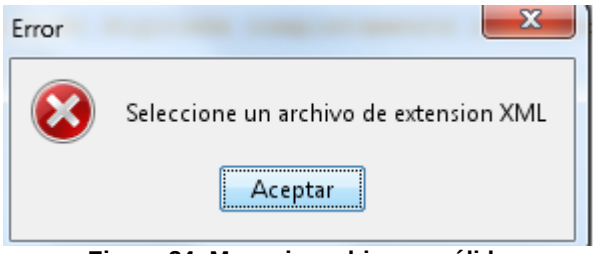


Figura 24: Mensaje archivo no válido

4.3.3.2 Falta definición de algún elemento en el archivo de entrada

Cuando se recibe un archivo de entrada XML con la consulta especificada, éste debe tener definido tanto los elementos fijos que no varían de un operador a otro (nodo "consulta", nodo "operador") como los elementos definidos para cada operador. Para probar esta funcionalidad se utilizó un archivo que no contiene el elemento "operador".

El archivo de entrada seleccionado (ver figura 25) contiene todos los elementos menos el tipo de operador. El sistema mostrará un mensaje notificando la no existencia de un elemento en el archivo (ver figura 26).

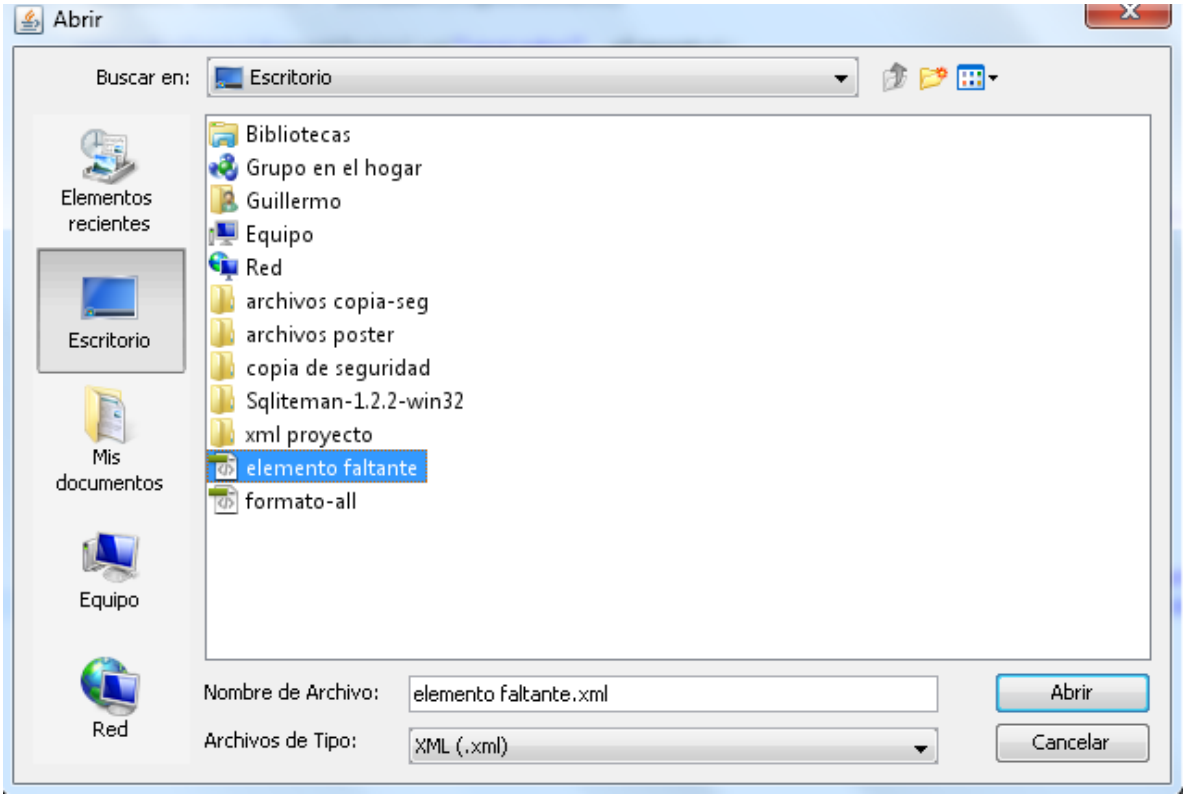


Figura 25: Selección de un archivo elemento faltante

Figura 26: Mensaje archivo elemento faltante

4.3.3.3 Elemento del archivo de entrada no tiene valor

En el archivo de entrada donde se especifica la consulta, debe contener todos los elementos definidos en la estructura para cada tipo de operador. Un elemento que requiera de un valor puede estar definido dentro del archivo y no tener valor alguno. Por lo que no se podrá seguir con la lectura del archivo. Para probar esta funcionalidad se usó un archivo que contiene el elemento operador que requiere de un valor y no lo tiene.

El archivo de entrada (ver figura 27) contiene todos los elementos definidos de acuerdo a la estructura del formato, pero el elemento “operador” no tiene un valor. El sistema debe mostrar un mensaje notificando de la falta de un valor en un elemento (ver figura 28).

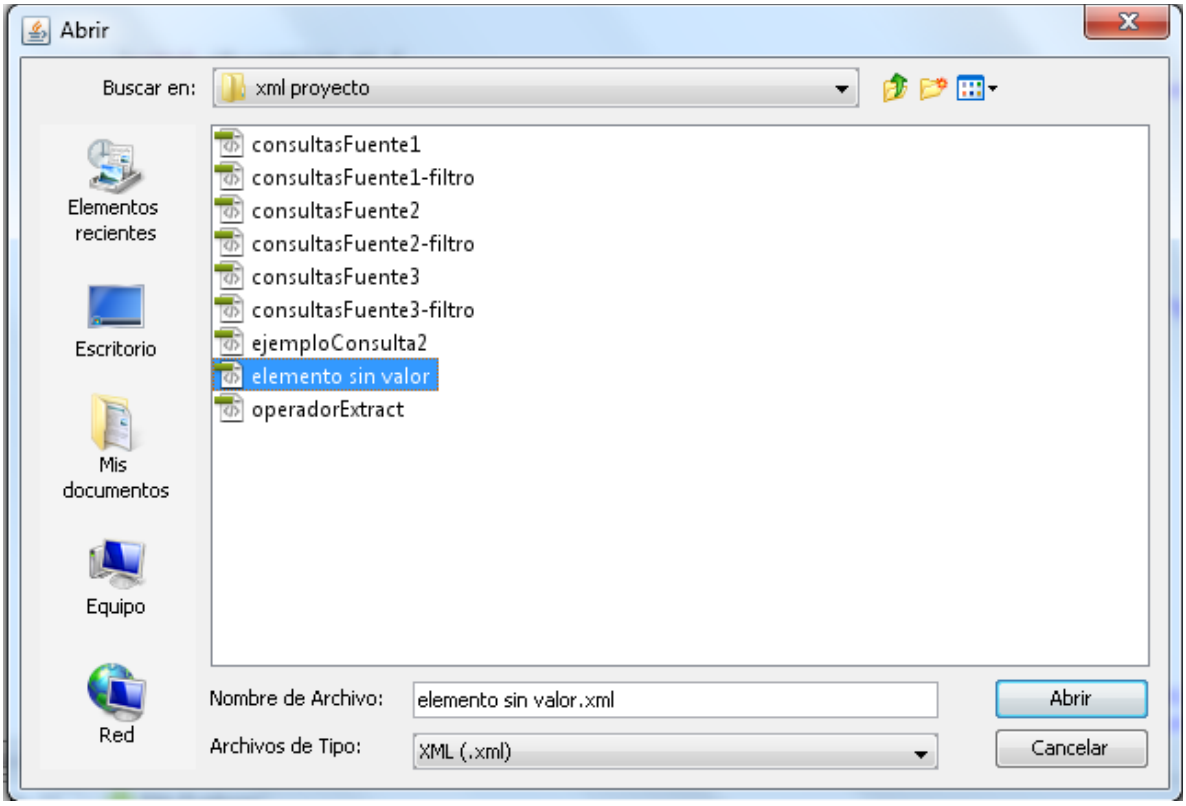


Figura 27: Selección de archivo elemento sin valor

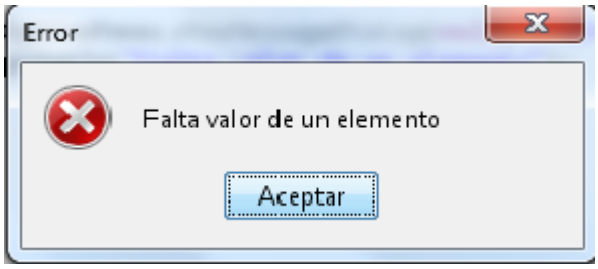


Figura 28: Selección de archive elemento sin valor

4.3.3.4 Consulta que involucra el operador extracción de subgrafos

La prueba de esta funcionalidad se hizo con 3 ejemplos de consulta. El primer ejemplo involucra dos nodos. El segundo ejemplo, descrito en la sección 4.2.4, involucra tres nodos. El último ejemplo involucra cuatro nodos. Estas consultas de prueba se realizaron sobre la implementación del modelo global de datos y la representación de mapeos definido en la sección 4.2. Cabe aclarar que el mediador funciona para cualquier modelo global definido en GDM.

Como resultado de la ejecución de cada una de las consultas se generan tres archivos que contienen las subconsultas para cada fuente. Estas subconsultas están definidas siguiendo la especificación del formato de salida. A continuación se describen cada una de las pruebas, haciendo énfasis en el segundo ejemplo. Los resultados de la ejecución

de estas consultas no se muestran en el formato de salida debido a la extensión de estos archivos.

4.3.3.4.1 Consulta que involucra dos nodos operador extracción de subgrafos

El archivo de entrada para esta consulta:

Extract({paLastName, diDescription })

es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<consulta>
<operador>EXTRACT</operador>
<nodosRespuesta>
<nodo>
<id>pLN</id>
</nodo>
<nodo>
<id>diD</id>
</nodo>
</nodosRespuesta>
</consulta>
```

Figura 29: Archivo de entrada consulta operador extracción de subgrafos con dos nodos involucrados

En el archivo de entrada, ”pLN” es el identificador en el esquema global del nodo paLastName y “diD” es el identificador en el esquema global del nodo diDescription. En esta consulta se busca encontrar los posibles caminos no dirigidos en el esquema global entre los nodos paLastName y diDescription, estos caminos conforman el esquema de respuesta. La instancia de respuesta incluye todos los nodos que pertenecen a las clases del esquema respuesta y los arcos entre ellos.

El esquema de respuesta para esta consulta está compuesto por los siguientes caminos:

{(Patient,lastName,paLastName),(Encounter,of,Patient),(Encounter,diagnosis,diCode),(Disease,diCode,diCode),(Disease,description,diDescription),(Encounter,familyHistory,ß1),(ß1,disease,diCode),(Encounter,pastMedicalHistory,ß2),(ß2,disease,diCode)}

La siguiente imagen muestra este esquema de repuesta representado en el modelo global de la figura 18:

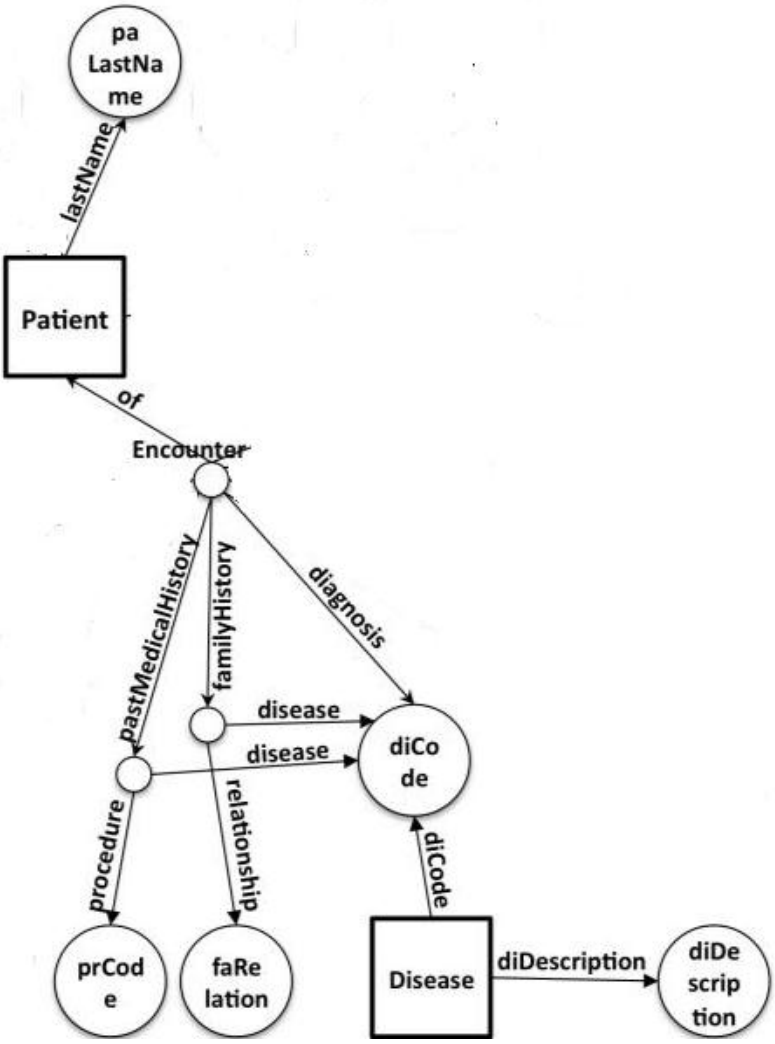


Figura 30: Subgrafo de Respuesta para Consulta de operador Extract que Involucra Dos Nodos.

Los resultados para esta consulta de prueba son:

Fuente S1

```
Select   ?paID ?paLastName
where { ?Patient paID ?paID . OPTIONAL{?Person  peID  ?paID . ?Person
peLastName  ?paLastName }}
```

Fuente S2

```
Select  ?paID ?enID
where { ?Encounter enID ?enID . ?Patient paID ?paID . ?Encounter  paID  ?paID }

Select   ?diCodeD ?diCodeFh ?diCodeMh ?enID
where {?Encounter enID ?enID . OPTIONAL{?Diagnosis diCode ?diCodeD . ?Diagnosis
enID ?enID }. OPTIONAL{?FamilyHistory  diCode  ?diCodeFh . ?FamilyHistory  enID
?enID }. OPTIONAL{?PastMedicalHistory  diCode  ?diCodeMh . ?PastMedicalHistory
enID  ?enID }}
```

```
Select   ?paID ?paLastName
where { ?Patient paID ?paID . OPTIONAL{?Patient  paLastName  ?paLastName }}

Select   ?diDescription ?diCode
where { ?Disease diCode ?diCode . OPTIONAL{?Disease  diDescription
?diDescription }}
```

Fuente S3

```
Select   ?diDescription ?diCode
where { ?Disease diCode ?diCode . OPTIONAL{?Disease  diDescription
?diDescription }}
```

4.3.3.4.2 Consulta que involucra tres nodos operador extracción de subgrafos

El archivo de entrada para la consulta

Extract({paLastName, *phLastName ,diDescription })*

es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<consulta>
<operador>EXTRACT</operador>
<nodosRespuesta>
<nodo>
<id>pLN</id>
</nodo>
<nodo>
<id>phLN</id>
</nodo>
<nodo>
<id>diD</id>
</nodo>
</nodosRespuesta>
</consulta>
```

Figura 31: Archivo de entrada consulta operador extracción de subgrafos con tres nodos involucrados

En el archivo de entrada, ”pLN” es el identificador en el esquema global del nodo paLastName, “phLN” es el identificador en el esquema global del nodo phLastName y “diD” es el identificador en el esquema global del nodo diDescription. En esta consulta se buscan en el esquema global los caminos no dirigidos entre las combinaciones posibles de nodos. Estos caminos conforman el esquema de respuesta. La instancia de respuesta incluye todos los nodos que pertenecen a las clases del esquema respuesta y los arcos entre ellos.

El esquema de respuesta para esta consulta está compuesto por los siguientes caminos:

{(Patient,lastName,paLastName),(Encounter,of,Patient),(Encounter,diagnosis,diCode),(Disease,diCode,diCode),(Disease,description,diDescription),(Encounter,familyHistory,ß1),(ß1,disease,diCode),(Encounter,pastMedicalHistory,ß2),(ß2,disease,diCode),(Physician,lastName,phLastName),(Encounter,attendedBy,Physician)}

La siguiente imagen muestra este esquema de repuesta representado en el modelo global de la figura 18:

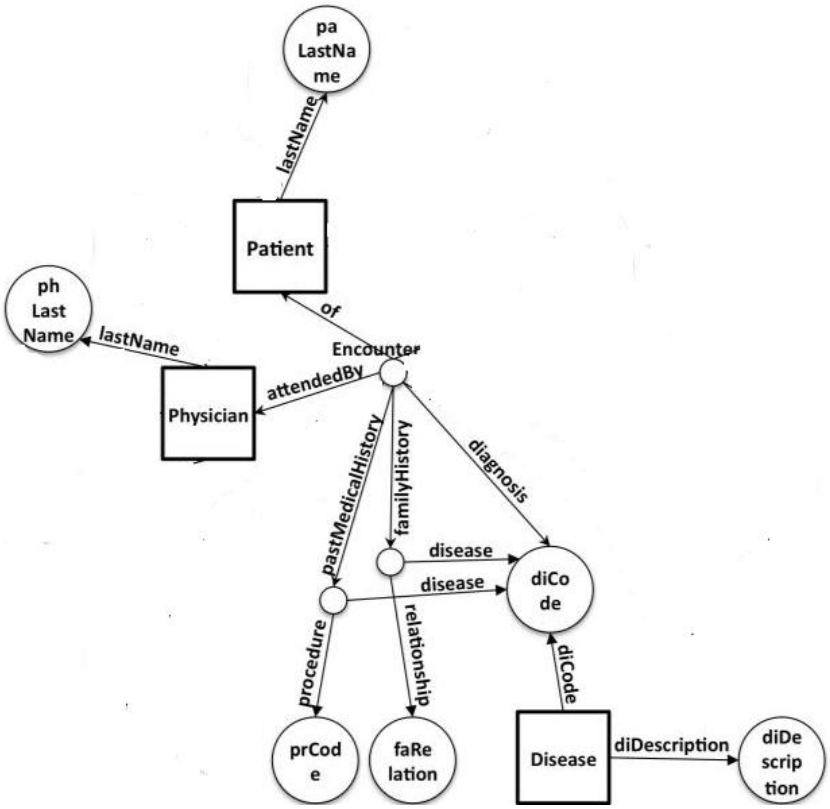


Figura 32: Subgrafo de Respuesta para Consulta de operador Extract que Involucra Tres Nodos.

En la figura 33 se muestra una subconsulta hacia la fuente S1 como resultado del procesamiento de la consulta.

```
<consulta>
<select>
<nodoRetorno>?paID</nodoRetorno>
<nodoRetorno>?paLastName</nodoRetorno>
</select>
<where>
<patron>
<arco>
<clase>?Patient</clase>
<relacion>paID</relacion>
<objeto>?paID</objeto>
</arco>
<optional>
<arco>
<clase>?Person</clase>
<relacion>peID</relacion>
<objeto>?paID</objeto>
</arco>
<arco>
<clase>?Person</clase>
<relacion>peLastName</relacion>
<objeto>?paLastName</objeto>
</arco>
</optional>
</patron>
</where>
</consulta>
```

Figura 33: Subconsulta hacia la fuente S1 operador extracción de subgrafos que involucra tres

Los resultados completos para esta consulta de prueba son:

Fuente S1

```
Select   ?phID ?phLastName
where { ?Physician phID ?phID . OPTIONAL{?Person  peID  ?phID . ?Person
peLastName  ?phLastName }}

Select   ?paID ?paLastName
where { ?Patient paID ?paID . OPTIONAL{?Person  peID  ?paID . ?Person
peLastName  ?paLastName }}
```

Fuente S2

```
Select   ?phID ?phLastName
where { ?Physician phID ?phID . OPTIONAL{?Physician  phLastName  ?phLastName }}

Select  ?phID ?enID
where { ?Encounter enID ?enID . ?Physician phID ?phID . ?Encounter  phID  ?phID
}

Select  ?paID ?enID
where { ?Encounter enID ?enID . ?Patient paID ?paID . ?Encounter  paID  ?paID }
```

```
Select   ?diCodeD ?diCodeFh ?diCodeMh ?enID
where {?Encounter enID ?enID . OPTIONAL{?Diagnosis diCode ?diCodeD . ?Diagnosis
enID ?enID }. OPTIONAL{?FamilyHistory  diCode  ?diCodeFh . ?FamilyHistory  enID
?enID }. OPTIONAL{?PastMedicalHistory  diCode  ?diCodeMh . ?PastMedicalHistory
enID  ?enID }}

Select   ?paID ?paLastName
where { ?Patient paID ?paID . OPTIONAL{?Patient  paLastName  ?paLastName }}
```

Fuente S3

```
Select   ?diDescription ?diCode
where { ?Disease diCode ?diCode . OPTIONAL{?Disease  diDescription
?diDescription }}
```

4.3.3.4.3 Consulta que involucra cuatro nodos operador extracción de subgrafos

El archivo de entrada de la consulta

Extract({paLastName, *diCode ,diDescription , phId})*

es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<consulta>
<operador>EXTRACT</operador>
<nodosRespuesta>
<nodo>
<id>pLN</id>
</nodo>
<nodo>
<id>ldC</id>
</nodo>
<nodo>
<id>diD</id>
</nodo>
<nodo>
<id>phI</id>
</nodo>
</nodosRespuesta>
</consulta>
```

Figura 34: Archivo de entrada consulta operador extracción de subgrafos con cuatro nodos involucrados

En el archivo de entrada, “pLN” es el identificador en el esquema global del nodo paLastName, ”1dC” es el identificador en el esquema global del nodo diCode, “diD” es el identificador en el esquema global del nodo diDescription y “phI” es el identificador del nodo phId. En esta consulta se buscan en el esquema global los caminos no dirigidos entre las combinaciones posibles de nodos. Estos caminos conforman el esquema de

respuesta. La instancia de respuesta incluye todos los nodos que pertenecen a las clases del esquema respuesta y los arcos entre ellos.

El esquema de respuesta para esta consulta está compuesto por los siguientes caminos:

{(Patient,lastName,paLastName),(Encounter,of,Patient),(Encounter,diagnosis,diCode),(Disease,diCode,diCode),(Disease,description,diDescription),(Encounter,familyHistory,ß1),(ß1,disease,diCode),(Encounter,pastMedicalHistory,ß2),(ß2,disease,diCode),(Physician,idNumber,phId),(Encounter,attendedBy,Physician)}

La siguiente imagen muestra este esquema de repuesta representado en el modelo global de la figura 18:

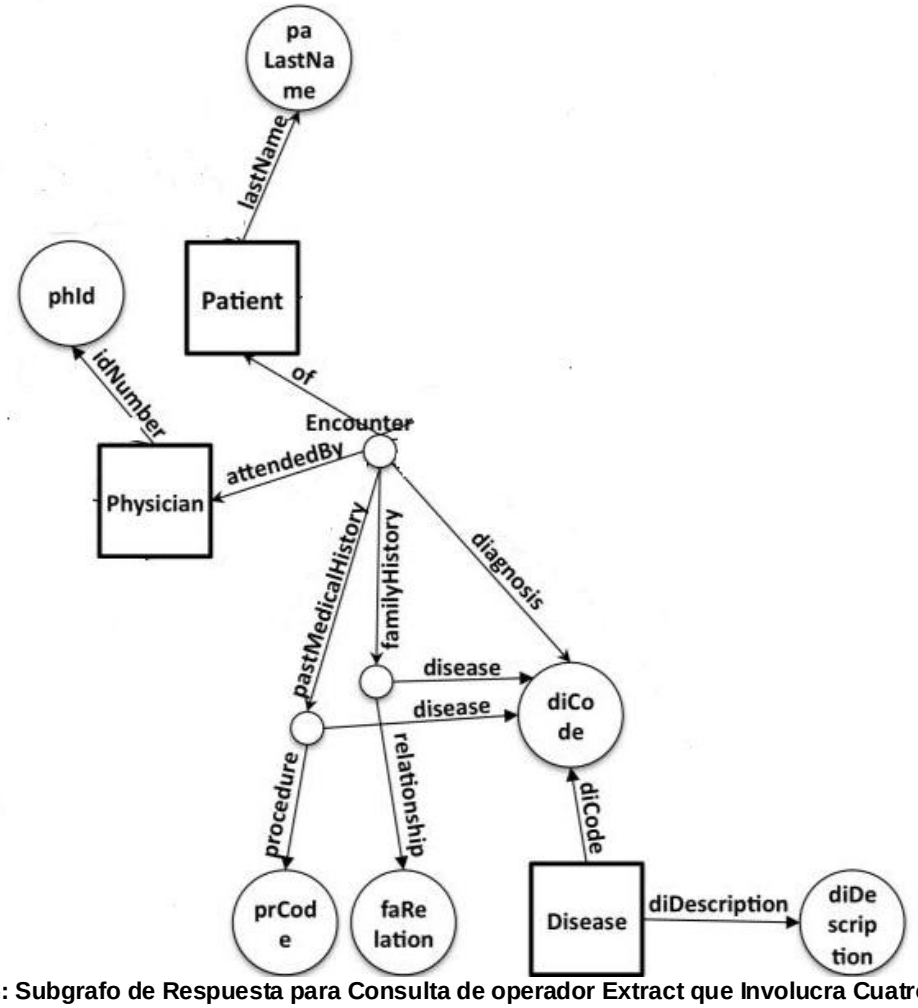


Figura 35: Subgrafo de Respuesta para Consulta de operador Extract que Involucra Cuatro Nodos.

Los resultados para esta consulta de prueba son:

Fuente S1

```
Select   ?paID ?paLastName
where { ?Patient paID ?paID . OPTIONAL{?Person  peID  ?paID . ?Person
peLastName  ?paLastName }}
```

Fuente S2

```
Select  ?phID ?enID
where { ?Encounter enID ?enID . ?Physician phID ?phID . ?Encounter  phID  ?phID
}

Select  ?paID ?enID
where { ?Encounter enID ?enID . ?Patient paID ?paID . ?Encounter  paID  ?paID }

Select   ?diCodeD ?diCodeFh ?diCodeMh ?enID
where {?Encounter enID ?enID . OPTIONAL{?Diagnosis diCode ?diCodeD . ?Diagnosis
enID ?enID }. OPTIONAL{?FamilyHistory  diCode  ?diCodeFh . ?FamilyHistory  enID
?enID }. OPTIONAL{?PastMedicalHistory  diCode  ?diCodeMh . ?PastMedicalHistory
enID  ?enID }}

Select   ?paID ?paLastName
where { ?Patient paID ?paID . OPTIONAL{?Patient  paLastName  ?paLastName }}
```

Fuente S3

```
Select   ?diDescription ?diCode
where { ?Disease diCode ?diCode . OPTIONAL{?Disease  diDescription
?diDescription }}
```

4.3.3.5 Consulta que involucra el operador filtro de valor

La prueba de esta funcionalidad se hizo con 3 ejemplos de consulta. El primer ejemplo, descrito en la sección 4.2.5, involucra dos nodos. El segundo ejemplo involucra tres nodos. El último ejemplo involucra cuatro nodos. Estas consultas de prueba se realizaron sobre la implementación del modelo global de datos y la representación de mapeos definido en la sección 4.2 al igual que para las pruebas del operador de extracción de subgrafos. A continuación se detalla el primer ejemplo. Para los otros dos ejemplos solo se nombran los nodos involucrados y la expresión de filtro.

Como resultado de la ejecución de estas consultas al igual que en el operador anterior se generan tres archivos que contienen las subconsultas para cada fuente. Estas subconsultas están definidas siguiendo la especificación del formato de salida. Los resultados de la ejecución de estas consultas no se muestran en el formato de salida debido a la extensión de estos archivos.

4.3.3.5.1 Consulta que involucra dos nodos

El archivo de entrada para la consulta:

ValueFilter ({paLastName$_{\{p1\}}$, enDate$_{\{p2\}}$}, diDescription = “Diabetes")

Donde:
{p1}={(Patient,lastName,paLastName),(Encounter,of,Patient),(Encounter,diagnosis,diCode),(Disease,diCode,diCode) (Disease,description,diDescription)},
{p2}={(Encounter,encounterDate,enDate),(Encounter,diagnosis,diCode),(Disease,diCode,diCode),(Disease,description,diDescription)}

es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<consulta>
<operador>VALUEFILTER</operador>
<nodosRespuesta>
<nodo>
<id>pLN</id>
</nodo>
<pk>
<arco>
<id>patln</id>
</arco>
<arco>
<id>enfp</id>
</arco>
<arco>
<id>endic</id>
</arco>
<arco>
<id>didco</id>
</arco>
<arco>
<id>disdi</id>
</arco>
</pk>
<nodo>
<id>cDate</id>
</nodo>
<pk>
<arco>
<id>endat</id>
</arco>
<arco>
<id>endic</id>
</arco>
<arco>
<id>didco</id>
</arco>
<arco>
<id>disdi</id>
</arco>
</pk>
</nodosRespuesta>
<filtro>
<condicion>
<nodo>diD</nodo>
<operador>=</operador>
<valor>diabetes</valor>
</condicion>
</filtro>
</consulta>
```

Figura 36: Archivo de entrada consulta filtro de valor con dos nodos involucrados

En el archivo de entrada, “pLN” es el identificador en el esquema global del nodo paLastName. Los identificadores de los arcos del camino p1 se muestran debajo del nodo en el orden respectivo. “cDate” es el identificador en el esquema global del nodo enDate. Los identificadores de los arcos del camino p2 se muestran debajo del nodo en el orden respectivo. “diD” que se encuentra dentro del nodo filtro es el identificador en el esquema

global del nodo diDescription. Esta consulta selecciona datos de valor básico de los caminos p1 y p2 que contengan el nodo “diDescription” y cuyo valor en la instancia de este nodo sea igual a “Diabetes”.

En la figura 37 se muestra una subconsulta hacia la fuente S3 como resultado del procesamiento de la consulta.

```
<consulta>
<select>
<nodoRetorno>?diDescription</nodoRetorno>
<nodoRetorno>?diCode</nodoRetorno>
</select>
<where>
<patron>
<arco>
<clase>?Disease</clase>
<relacion>diCode</relacion>
<objeto>?diCode</objeto>
</arco>
<arco>
<clase>?Disease</clase>
<relacion>diDescription</relacion>
<objeto>?diDescription</objeto>
</arco>
</patron>
<filtro>
<expresion>
<dato>?diDescription</dato>
<operadorComparacion>=</operadorComparacion>
<valor>diabetes</valor>
</expresion>
</filtro>
</where>
</consulta>
```

Figura 37: Subconsulta hacia la fuente S3 operador filtro de valor que involucra dos nodos.

Los resultados completos para esta consulta de prueba son:

Fuente S1

```
Select   ?paID ?paLastName
where { ?Patient paID ?paID . ?Person  peID  ?paID . ?Person  peLastName
?paLastName }
```

Fuente S2

```
Select   ?paID ?paLastName
where { ?Patient paID ?paID . ?Patient  paLastName  ?paLastName }

Select  ?paID ?enID
El where { ?Encounter enID ?enID . ?Patient paID ?paID . ?Encounter  paID  ?paID
}
```

```
Select   ?diCodeD ?enID
where { ?Encounter enID ?enID . ?Diagnosis diCode ?diCodeD . ?Diagnosis enID
?enID }

Select   ?diCodeD ?enDate ?enID
where {?Encounter enID ?enID . ?Encounter encounterDate ?enDate . ?Diagnosis
diCode ?diCodeD . ?Diagnosis enID ?enID}
```

Fuente S3

```
Select   ?diDescription ?diCode
where { ?Disease diCode ?diCode . ?Disease  diDescription  ?diDescription  .
FILTER(?diDescription="diabetes")}

Select   ?diDescription ?diCode
where { ?Disease diCode ?diCode . ?Disease  diDescription  ?diDescription  .
FILTER(?diDescription="diabetes")}
```

4.3.3.5.2 Consulta que involucra tres nodos

La consulta es expresada como:

> *ValueFilter* ({phId$_{\{p1\}}$,patLastName$_{\{p2\},}$ pathBirthDate$_{\{p3\}}$}, diDescription = “Cancer de pulmon")
> Donde:
> **{p1}**={(Physician,idNumber,phId),(Encounter,attendedBy,Physician),(Encounter,diagnosis,diCode),(Disease,diCode,diCode) (Disease,description,diDescription)},
> **{p2}**={(Patient,lastName,paLastName),(Encounter,of,Patient),(Encounter,diagnosis,diCode),(Disease,diCode,diCode) (Disease,description,diDescription)},
> **{p3}**={(ß1,dateOfBirth,patBirthDay),(Patient, demographics, ß1),(Encounter,of,Patient),(Encounter,diagnosis,diCode),(Disease,diCode,diCode) (Disease,description,diDescription)}

El archivo de entrada de esta consulta contiene los identificadores en el esquema global con la misma estructura que el archivo de la figura 32 de cada uno de los nodos, y debajo de estos los identificadores de los arcos del camino correspondiente al nodo. Esta consulta selecciona datos de valor básico de los caminos p1, p2 y p3 que contengan el nodo “diDescription” y cuyo valor en la instancia de este nodo sea igual a “Cancer de pulmon”.

Los resultados para esta consulta de prueba son:

Fuente S1

```
Select   ?paID ?paLastName
where { ?Patient paID ?paID . ?Person  peID  ?paID . ?Person  peLastName
?paLastName }

Select   ?paID ?paBirthDate
where { ?Patient paID ?paID . ?Patient  paBirthDate  ?paBirthDate }
```

Fuente S2

```
Select  ?phID ?enID
where { ?Encounter enID ?enID . ?Physician phID ?phID . ?Encounter  phID  ?phID
}

Select   ?diCodeD ?enID
where { ?Encounter enID ?enID . ?Diagnosis diCode ?diCodeD . ?Diagnosis enID
?enID }

Select   ?paID ?paLastName
where { ?Patient paID ?paID . ?Patient  paLastName  ?paLastName }

Select  ?paID ?enID
where { ?Encounter enID ?enID . ?Patient paID ?paID . ?Encounter  paID  ?paID }

Select   ?diCodeD ?enID
where { ?Encounter enID ?enID . ?Diagnosis diCode ?diCodeD . ?Diagnosis enID
?enID }

Select  ?paID ?enID
where { ?Encounter enID ?enID . ?Patient paID ?paID . ?Encounter  paID  ?paID }

Select   ?diCodeD ?enID
where { ?Encounter enID ?enID . ?Diagnosis diCode ?diCodeD . ?Diagnosis enID
?enID }
```

Fuente S3

```
Select   ?diDescription ?diCode
where { ?Disease diCode ?diCode . ?Disease  diDescription  ?diDescription  .
FILTER(?diDescription="cancer de pulmon")}

Select   ?diDescription ?diCode
where { ?Disease diCode ?diCode . ?Disease  diDescription  ?diDescription  .
FILTER(?diDescription="cancer de pulmon")}

Select   ?diDescription ?diCode
where { ?Disease diCode ?diCode . ?Disease  diDescription  ?diDescription  .
FILTER(?diDescription="cancer de pulmon")}
```

Para este caso de prueba se aprecia que hay subconsultas repetidas para la Fuente S2 y para la Fuente S3. Esto sucede debido a que las subconsultas para este operador son generadas por cada camino definido y estos pueden tener arcos comunes((Encounter,of,Patient),(Encounter,diagnosis,diCode),(Disease,diCode,diCode) (Disease,description,diDescription)) como ocurre en este caso. Esto da como resultado que una subconsulta pueda ser generada más de una vez para una misma fuente.

4.3.3.5.3 Consulta que involucra cuatro nodos

La consulta es expresada como:

ValueFilter ({paLastName $_{p1}$},paFirstName$_{\{p2\}}$, phId$_{\{p3\}}$, enType$_{\{p3\}}$}, patGender = "Masculino")
Donde:
{p1}={(Patient,lastName,paLastName),(Patient,demographics, ß1),(ß1,gender,patGender)},
{p2}={(Patient,firstName,patFirstName),(Patient,demographics, ß1),(ß1,gender,patGender)},
{p3}=(Physician,idNumber,phId),(Encounter,attendedBy,Physician) ,(Encounter,of,Patient),(Patient,demographics, ß1),(ß1,gender,patGender)},
{p4}={(Encounter,encounterType,enType),(Encounter,of,Patient)(Patient,demographics, ß1),(ß1,gender,patGender)}

El archivo de entrada de esta consulta contiene los identificadores en el esquema global con la misma estructura que el archivo de la figura 32 de cada uno de los nodos, y debajo de estos los identificadores de los arcos del camino correspondiente al nodo. Esta consulta selecciona datos de valor básico de los caminos p1, p2, p3 y p4 que contengan el nodo "patGender" y cuyo valor en la instancia de este nodo sea igual a "Masculino".

Los resultados para esta consulta de prueba son:

Fuente S1

```
Select   ?paGender ?paID ?paLastName
where {?Patient paID ?paID . ?Person  peID  ?paID . ?Person  peLastName
?paLastName . ?Patient  paGender  ?paGender  . FILTER(?paGender="masculino")}

Select   ?paFirstName ?paGender ?paID
where {?Patient paID ?paID . ?Person  peFirstName  ?paFirstName . ?Person  peID
?paID . ?Patient  paGender  ?paGender  . FILTER(?paGender="masculino")}

Select   ?paGender ?paID
where { ?Patient paID ?paID . ?Patient  paGender  ?paGender  .
FILTER(?paGender="masculino")}

Select   ?paGender ?paID
where { ?Patient paID ?paID . ?Patient  paGender  ?paGender  .
FILTER(?paGender="masculino")}
```

Fuente S2

```
Select   ?paID ?paLastName
where { ?Patient paID ?paID . ?Patient  paLastName  ?paLastName }

Select   ?paFirstName ?paID
where { ?Patient paID ?paID . ?Patient  paFirstName  ?paFirstName }

Select  ?phID ?enID
where { ?Encounter enID ?enID . ?Physician phID ?phID . ?Encounter  phID  ?phID
}

Select  ?paID ?enID
```

```
where { ?Encounter enID ?enID . ?Patient paID ?paID . ?Encounter  paID  ?paID }

Select   ?enType ?enCode ?enID
where { ?Encounter enID ?enID . ?Encounter  enCode ?enCode . ?EncounterType
enCode  ?enCode . ?EncounterType  enType  ?enType }
Select  ?paID ?enID
where { ?Encounter enID ?enID . ?Patient paID ?paID . ?Encounter  paID  ?paID }
```

Fuente S3

No hay consultas para esta Fuente.

Para este caso de prueba al igual que el anterior se tienen subconsultas repetidas, en este caso para la Fuente S1 y la Fuente S2. Esto se debe a que los caminos definidos para esta consulta tiene arcos comunes((Patient,demographics, ß1),(ß1,gender,patGender),(Encounter,of,Patient)).Como este operador genera las suboconsultas por cada camino esto da como resultado que un subconsulta sea generada más de una vez para una misma fuente.

Capítulo 5

En este capítulo se discuten los resultados del trabajo y se proponen algunas líneas de trabajo futuro.

El propósito de este trabajo era desarrollar un prototipo funcional de mediador, que hace parte de una arquitectura de mediación propuesta en “Lenguaje para la Recuperación de Datos Heterogéneos en el Dominio Médico” [28].

Para hacerlo fue necesaria la revisión de literatura de proyectos similares para buscar soluciones ya propuestas y tomar de allí bases para el proyecto. Dentro de los proyectos revisados se encuentran [5], [6], [7], [8],[14], [15], [16] y [17] que se detallan en la sección 2.2 de este documento.

Ninguno de los proyectos revisados implementaba una arquitectura de integración con un esquema global orientado a grafos. Por lo tanto, ninguna de las característica de estos se tomó como base para el proyecto. Sin embargo, la revisión de estos trabajos sirvió para definir líneas futuras en el proyecto desarrollado. Por ejemplo, incluir técnicas para optimizar el plan de ejecución global de la consulta, como se hace en [5] y en [6].

El prototipo desarrollado representa la información de las fuentes mediante un modelo de datos basado en grafos. Este, es un modelo conceptual del dominio de la aplicación y está formado por un conjunto de subgrafos que representa la información contenida en cada fuente de datos.

Una de las tecnologías utilizadas para el desarrollo del prototipo fue *Neo4j,* una base de datos basada en grafos que facilita la representación y almacenamiento del modelo de datos del mediador. La selección de *Neo4j* implicó hacer una revisión de la documentación de otras bases de datos orientadas a grafos existentes en el mercado como InfiniteGraph, AllegroGraph, OrientDB, descritas en la sección 2.1.4. También dentro del proceso de revisión se pensó hacer uso del modelo implementado en [29], pero esta opción se descartó principalmente porque el proyecto ya había finalizado. Lo que implicaba tener documentación desactualizada y hacer uso de una herramienta sin soporte.

Uno de los requerimientos del proyecto era implementar un modelo de datos GDM para representar el modelo global del mediador. En este sentido, fue necesario adaptar el modelo de grafos ofrecido por Neo4j (Grafo atribuido) para que se pudiera representar el modelo GDM sin que afectara su definición, propiedades y para que a la hora de realizar el procesamiento de las consultas los resultados no se vieran afectados Esto significó un proceso de estudio del modelo de Neo4j en profundidad y de la funcionalidad embebida de esta bases de datos sobre el lenguaje JAVA. Además de un estudio del modelo GDM, para definir una correspondencia entre los elementos del grafo GDM y del grafo de *Neo4j.*

El mediador desarrollado recibe la consulta en formato XML. A partir del operador identificado en el formato se generan las subconsultas.

Dos operadores se implementaron, extracción de subgrafos y filtro de valor. El operador de extracción de subgrafos tiene como entrada un conjunto de nodos de interés del esquema (modelo Global). Este operador busca en el esquema global los caminos no dirigidos entre los nodos de interés para construir con ellos el esquema de respuesta. El esquema de respuesta resulta de la unión de los caminos hallados. La instancia de respuesta incluye todos los nodos que pertenecen a las clases del esquema respuesta y los arcos entre ellos. El operador filtro tiene como entrada un conjunto de nodos de interés del esquema y una expresión de la forma *mb op v* (*m* es un nodo del esquema, *op* es un operador de comparación y *v* un valor). Cada nodo de interés tienen asociado un camino que lo une con el nodo de la expresión de valor. Este operador busca seleccionar datos de valor básico que están conectados por un camino con un nodo de valor básico cuya instancia cumple con la condición especificada en la expresión de filtro.

Cuando se aplica el operador extracción de subgrafos, se identifican los nodos de respuesta en el esquema global por medio de los identificadores. Estos identificadores vienen definidos en el archivo de la consulta. Se halla el subgrafo esquema a partir de estos nodos y se procede a generar las subconsultas sobre las fuentes donde hay información de los nodos y arcos del subgrafo esquema. Las fuentes que poseen información de los nodos y arcos, son las fuentes donde tiene mapeos los arcos del subgrafo esquema.

Si el operador es filtro de valor se identifican: los nodos de respuesta en esquema global por medio de los identificadores, cada uno de los caminos de los nodos de respuesta al nodo de la expresión de filtro y la expresión de filtro y se generan las subconsultas a las fuentes donde hay información de los nodos y arcos de cada camino. Las fuentes que poseen información de los nodos y arcos, son las fuentes donde tiene mapeos los arcos del subgrafo esquema. El proceso anteriormente detallado y la definición de los operadores implementados siguen la especificación dada en [30]. Las subconsultas se generan en un formato XML común para los dos operadores. Por cada fuente se genera un archivo con las consultas correspondientes a la fuente.

Algunas conclusiones derivadas de este trabajo son las siguientes:

1. Se ha conseguido desarrollar un prototipo de mediador que identifica en cuáles fuentes de datos existe información para dar respuesta a una consulta de un usuario.

2. El prototipo de mediador descompone una consulta de usuario en subconsultas sobre cada una de las fuentes de datos que contienen información relevante para la consulta (fuentes donde hay información para dar respuesta a una consulta).

3. El prototipo de mediador desarrollado funciona independientemente de las fuentes de datos. Para efectos del desarrollo no se estableció conexión con las fuentes de datos.

4. El prototipo es que es independiente del dominio de aplicación es decir, funciona para cualquier modelo global definido en GDM.

5. Finalmente, a nivel personal, la realización de este trabajo de fin de carrera ha sido una experiencia muy positiva, tanto por los objetivos marcados los cuales eran novedosos para los conocimientos que se adquirieron durante la carrera, como por los problemas que se encontraron durante la realización del proyecto y las soluciones que se aplicaron para superarlos.

Algunas líneas de trabajo futuro que se podrían derivar del trabajo podrían ser:

- Ampliar el número de operadores implementados.
- Incluir técnicas para optimizar el plan de ejecución global de la consulta, como las propuestas en [5] y en [6].
- Como caso particular también se puede incluir optimización en el proceso generación de subconsultas para el operador de filtro de valor, para que no se generen subconsultas repetidas para una misma fuente cuando se tengan caminos con arcos comunes.
- Incluir composición de operadores. Es decir que una misma consulta pueda contener más de un tipo de operación.
- Incluir en la implementación del mediador el módulo de comunicación con las fuentes y la integración de las respuestas de las subconsultas. Al establecer una comunicación con las fuentes se obtendrán los resultados de las consultas representados en tripletas RDF. Estos datos se deben integrar por medio de los atributos identificadores y entregárselos al usuario para su visualización.

Referencias

[1]. Millán M., Pabón C., Una Arquitectura de Integración de Datos como Base de Lenguaje Conceptual de Recuperación de Datos Médicos. CLEI. Quito, Ecuador.2011.

[2].Vidal María. Integración de Datos en el Internet. [Artículo tomado de Internet]. http://www.essi.upc.edu/~www-si/seminari/abstrSeminari/MEVidal.pdf. [Febrero 23 2012].

[3]. Friedman, M., Levy, A., Millstein, T.: Navigational Plans for Data Integration.In: Proc. of the 16th National conf. on Artificial Intelligence and the 11th Conf.Innovative Applications of Artificial Intelligence AAAI'99/IAAI'99, pp. 67-73.American Association for Artificial Intelligence (1999).

[4].NationalElectrical Manufactures Association. Digital Imaging and Communication in Medicine (DICOM). Rssylin, VA: NEMA.

[5].Tomasic A, Amourou R, Bonnet P, Kapitskaia O, Naacke H, Raschid L. The Distributed Information Search Component (Disco) and the World Wide Web. [Artículo tomado de Internet]. http://www.cs.cmu.edu/~tomasic/doc/1997/TomasicEtAlSIGMODDEMO1997.pdf [Consulta: Abril 3 de 2012].

[6]. Zachary G. Ives, Daniela Florescu, Marc Friedman, Alon Levy, Daniel S. Weld. An Adaptive Query Execution System for Data Integration. SIGMOD '99 Proceedings of the 1999 ACM SIGMOD international conference on Management of data. 1999: 299-310 . New York. USA.

[7]. Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer,Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullman, Jennifer Widom. The TSIMMIS Project: Integration of Heterogeneous Information Sources. In 16th Meeting of the Information Processing Society of Japan (1994).

[8]. Sanchis, J.A. Maldonado, P.Crespo, M. Robles, R. Garcia , M.Correcher. MIDAS: Un sistema para el acceso unificado a registros clínicos distribuidos.[Artículo tomado de Internet]. http://www.seis.es/seis/inforsalud04/sanchisa.htm. [Consulta: Abril 3 de 2012].

[9]. Palacios Escalona, juan pablo Y Costilla Rodríguez, Carmen. Modelo de Unificación Semántica de Ontologías, aplicado al dominio de los archivos digitales. Tesis Doctoral UPM-Departamento de ingeniería de sistemas telemáticos (2005).

[10]. Batini, C. Scannapieco, M. Data Quality: Concepts, Methodologies and Techniques. Berlin, Springer-VerlagBerlinHeidelberg(2006).

[11]. [Tomado de Internet].http://en.wikipedia.org/wiki/Ad_hoc. [Consulta: Junio 20 de 2012].

[12]. [Tomado de Internet].http://en.wikipedia.org/wiki/Graph_database.[Consulta: Octubre

26 de 2012].

[13]. W3C. SPARQL Query Languague for RDF. W3C Recommendation 15 January 2008.

[14]. Hernandez C. Alejandro. Integración de fuentes de datos espaciales con base en ontologías [Tesis tomada de Internet]. http://itzamna.bnct.ipn.mx:8080/dspace/bitstream/123456789/9490/1/50.pdf. [Consulta: Octubre 26 de 2012].

[15]. Botello C. Alejandro. Explotación de Bases de Datos Heterogéneas Mediante Su Integración Parcial. [Articulo tomado de Internet]. http://www.gelbukh.com/enc2006/Trabajos/15_Alejandro%20Botella.pdf. [Consulta: Octubre 26 de 2012].

[16]. [Tomado de Internet]. http://virtuoso.openlinksw.com/virtual-database/.[Consulta: Octubre 26 de 2012].

[17]. Rick F. van der Lans. Developing a Data Delivery Platform With Informatica Data Services. [Articulo tomado de Internet]. http://www.r20.nl/Whitepaper%20Informatica%20The%20DDP%20Summary%20February%202011.pdf. [Consulta: Diciembre 15 de 2012].

[18]. Atul Adya, José A. Blakeley, Sergey Melnik, S. Muralidhar, and the ADO.NET Team. Anatomy of the ADO.NET Entity Framework. [Articulo tomado de Internet]. http://www.yamod.ch/media/6243/adya01.pdf . [Consulta: Diciembre 20 de 2012].

[19]. [Tomado de Internet]. http://www.neo4j.org/.[Consulta: Octubre 26 de 2012].

[20]. [Tomado de Internet]. http://en.wikipedia.org/wiki/InfiniteGraph.[Consulta: Octubre 26 de 2012].

[21]. [Tomado de Internet]. http://objectivity.com/INFINITEGRAPH.[Consulta: Octubre 26 de 2012].

[22]. [Tomado de Internet]. http://www.franz.com/agraph/allegrograph/.[Consulta: Octubre 26 de 2012].

[23]. [Tomado de Internet]. http://skos.um.es/TR/rdf-sparql-query/#describe [Consulta: Octubre 26 de 2012].

[24]. Philippe Kruchten. *The Rational Unified Process: An Introduction, Second Edition*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2000.

[25]. Amaro Calderón, Sarah Dámaris. Valverde Rebaza.,Jorge Carlos. Metodologías Agiles. Universidad Nacional de Trujillo, Facultad de Ciencias Físicas y Matemáticas. *Escuela de Informática.* [Tomado de Internet]. *http://www.seccperu.org/files/Metodologias%20Agiles.pdf* Consulta: Marzo 21 de 2013].

[26]. [Tomado de Internet]. http://www.mastersoft.com.ar/MsWeb/otros_archivos/NotaScrumPCUsers.pdf Consulta: Marzo 21 de 2013].

[27]. Renzo Angles, Claudio Gutierrez. *Survey of graph database models*. *Journal ACM Computing Surveys (CSUR), Volume 40 Issue 1, February 2008*.

[28]. María C. Pabón. Lenguaje de Recuperación de Datos Heterogéneos en el Dominio Médico. Tesis Doctoral. Escuela de Ingeniería de Sistemas y Computación. Universidad del Valle. Junio de 2011.

[29]. Yigal Arens, Chun-Nan Hsu, Craig A. Knoblock. *Query processing in the SIMS information mediator. Published in: Book Readings in agents. Pages 82-90. San Francisco, CA, USA.1998.*

[30] María C. Pabón. Definición de la Arquitectura de Mediación para la Integración de Datos Heterogéneos en el Domino Médico. Universidad del Valle. Abril de 2013.

[31] María C. Pabón , Martha Millán, C. Roncancio. Graph Data Transformations and Querying. Submitted for publication, 2013.

ANEXOS

ANEXO A: Especificación de requerimientos

	Especificación de Requerimientos ISO-9000 Universidad del Valle	**Documento:** ER-001	**Revisión:** 001
Especificación de Requerimientos de la aplicación		**Página:** 1/2 **Fecha:** 21/03/2013	

1. Módulo: Consultas.

Req.	Descripción
R1.1	El sistema en el módulo de consultas, recibe la consulta proveniente de una interfaz de usuario e identifica qué tipo de operador involucra la consulta y que parámetros contiene.
R1.2	El sistema en el módulo de consultas, debe identificar cuáles nodos se retornan como respuesta y cuáles arcos forman parte del patrón de búsqueda dependiendo del tipo de operador definido en la consulta (filtro y subgrafo).
R1.3	El sistema en el módulo de consultas, debe identificar cuáles nodos tienen un filtro si el operador de la consulta es de tipo filtro.
R1.4	El sistema en el módulo de consultas, debe identificar cuáles fuentes de datos están involucradas en la consulta de un usuario.
R1.5	El sistema en el módulo de consultas, debe dividir la consulta realizada en subconsultas que cada fuente está en capacidad de responder
R1.6	El sistema en el módulo de consultas, debe reformular las subconsultas para que las entidades del modelo global se mapeen a entidades equivalentes definidas en el modelo local de la fuente de datos.

	Especificación de Requerimientos ISO-9000 Universidad del Valle	**Documento:** ER-002	**Revisión:** 001
Especificación de Requerimientos de la aplicación		**Página:** 2/2 **Fecha:** 21/03/2013	

2. Módulo: Comunicación.

Req.	Descripción
R2.1	El sistema en el módulo de comunicación, debe tener registro de las fuentes de datos. La información almacenada por cada fuente de datos es: id, nombre, URL y descripción.
R2.2	El sistema en el módulo de comunicación, debe establecer la comunicación con las fuentes de datos para enviar las consultas del usuario.
R2.3	El sistema en el módulo de comunicación, debe establecer una comunicación con las fuentes de datos para recibir las respuestas de la ejecución de las consultas

	Especificación de Requerimientos ISO-9000 Universidad del Valle	**Documento:** ER-003	**Revisión:** 001
Especificación de Requerimientos de la aplicación		**Página:** 2/2 **Fecha:** 21/03/2013	

3. Módulo: Integración de Respuestas.

Req.	Descripción
R3.1	El sistema en el módulo de integración de respuestas, debe integrar las respuestas de la ejecución de las consultas en cada fuente de datos. Los datos retornados de las fuentes se deben integrar por medio de los sus atributos identificadores y entregárselos al usuario para su visualización y entregárselas a la interfaz de usuario para su visualización.

ANEXO B: Documento de Product Backlog

	*Product Backlog*	**Documento: PB-001**	**Revision:001**	**Fecha: 21/03/2013**
Modulo: Consultas				
ID	**Título**	**Descripción**	**Prueba de Aceptación**	**Prioridad**
HC01	COMO usuario quiero que el sistema identifique que fuentes están involucradas en una consulta	Al sistema llega una consulta proveniente del usuario y este identifica que fuentes están involucradas.	Recibir una consulta y que el sistema identifique que fuentes responden a esa consulta	8
HC02	COMO usuario quiero que el sistema descomponga la consulta en subconsultas a las fuentes que contienen información relevante	Al sistema llega una consulta proveniente del usuario, a partir de la identificación de fuentes este descompone la consulta en subconsultas a las fuentes.	Recibir una consulta y que el sistema descomponga la consultas a subconsultas a las fuentes de datos involucradas	7
HC03	COMO usuario quiero que el sistema de respuesta a dos tipos de operadores involucrados en una consulta	Al sistema llega una consulta proveniente del usuario, este debe identificar que operador contiene la consulta para otorgar una respuesta de acuerdo a esto.	Recibir una consulta y que el sistema identifique el tipo de operador involucrado	9

	Product Backlog	**Documento: PB-002**	**Revision:001**	**Fecha: 21/03/2013**
Modulo: Comunicación				
ID	**Título**	**Descripción**	**Prueba de Aceptación**	**Prioridad**
HCM01	COMO usuario quiero que el sistema establezca comunicación con las fuentes de datos para enviar las consultas de usuario	El sistema genera una subconsulta hacia una fuente de datos, este debe establecer una comunicación con la fuente para enviar consulta.	A partir de una subconsulta generada el sistema la envía a la fuente de datos respectiva.	2
HCM02	COMO usuario quiero que el sistema establezca comunicación con las fuentes de datos para recibir las respuestas de la ejecución de las consultas	El sistema después de enviar una subconsulta a la fuente debe establecer una comunicación para recibir la respuesta.	A partir del envío de una subconsulta el sistema recibe la respuesta de la ejecución de esa en consulta en la fuente de datos respectiva.	1

	Product Backlog	**Documento: PB-003**	**Revision:001**	**Fecha: 21/03/2013**
Modulo: Integración				
ID	**Título**	**Descripción**	**Prueba de Aceptación**	**Prioridad**
HI01	COMO usuario quiero que el sistema integre las respuestas de la ejecución de las consultas y las entregue al usuario para su visualización.	El sistema después de la ejecución de las subconsultas en las fuentes debe integrar las respuestas para entregárselas al usuario	A partir de una consulta recibida del usuario el sistema enviara una respuesta unificada como resultado de la ejecución de las subconsultas en las fuentes	5

ANEXO C: Documento de Iteraciones

En este documento de iteraciones no se consideraron las historias de usuario del módulo de comunicación por estar fuera del alcance del proyecto.

	***Sprint 1:*identificación de consulta**	**Documento: SP-001**	**Revision:001**	**Inicio:04/02/13 Fin:03/03/13**
Pila del Sprint				
ID	**Tarea**	**Prioridad**	**Duración**	**Responsable**
HC03	Definir formato de consultas, identificación del tipo de operador, parámetros de consulta, nodos de respuesta y el patrón de consulta.	9	4 semanas	Equipo de Desarrollo

	***Sprint 2:*identificacion de fuentes y creación de subconsultas**	**Documento: SP-002**	**Revision:001**	**Inicio:04/03/13 Fin:31/03/13**
Pila del Sprint				
ID	**Tarea**	**Prioridad**	**Duración**	**Responsable**
HC02	Identificación de fuentes para cada arco dependiendo del tipo de operador, realización de mapeos.	8	2 semanas	Equipo de Desarrollo
HC01	Construcción de subconsultas a partir de la identificación de las fuentes y los mapeos	7	2 semanas	Equipo de Desarrollo

	Sprint 3: **Integración de resultados**	**Documento: SP-003**	**Revision:001**	**Fecha de Inicio:01/04/13 Fin: 28/04/13**
Pila del Sprint				
ID	**Tarea**	**Prioridad**	**Duración**	**Responsable**
HI01	Identificar los nodos integradores en cada subconsulta, generar el mapeo para la integración, incluirlo en la subconsulta correspondiente. Esto se realiza de acuerdo al tipo de operador	5	4 semanas	Equipo de Desarrollo

ANEXO D: Configuración y Puesta en Marcha de la Aplicación

Paquetería y clases

El código fuente de la aplicación se compone de los siguientes paquetes:

- conexionBD
- consultas
- integracion
- mediador

1. conexionBD: En este paquete se encuentra la clase que permite la conexión a la base de datos donde está almacenada la representación de mapeos y los mapeos de integración. Esta base de datos tiene como nombre ”mapeos”. Para que la aplicación se conecte a esta base de datos esta debe estar almacenada en un carpeta “BD mapeos”, que se encuentra en la carpeta raíz del proyecto. Es decir la ruta de la base de datos de mapeos debe ser “BD mapeos/mapeos”. La clase que componen este paquete es la siguiente:

- **ConexionBD.java:** Esta clase contiene los métodos que permiten acceder a los datos de la base de datos “mapeos”.

2. consultas: En este paquete se encuentra las clases que realizan el procesamiento de las consultas. Estas clases reciben las consultas en el formato XML respectivo y hacen uso de la base de datos “mapeos”, y del modelo global en *Neo4j* para generar las subconsultas a las fuentes. Las clases que componen este paquete son las siguientes:

- **Camino.java:** Esta clase permite representar los caminos expresados en las consultas.
- **Consulta.java:** Esta clase se encarga de realizar todo el procesamiento de consultas para los dos tipos de operadores. Esta clase hace uso de las demás de clase de este paquete para realizar esta función.
- **Fuente.java:** Esta clase permite representar las fuentes para las que se generan las consultas.
- **GeneradorXML.java:** Esta clase es la encargada de generar las consultas en el formato de salida especificado.
- **Optional.java:** Esta clase permite representar las triplas que hacen parte de un “OPTIONAL” para las consultar generadas con el operador extracción de subgrafos.
- **Subconsulta.java:** Esta clase permite representar las subconsultas generadas hacia las fuentes.
- **Tripla.java:** Esta clase permite representar las triplas del modelo global.
- **TriplaModeloLocal.java:** Esta clase permite representar las triplas a las que son mapeadas las triplas del modelo global.
- **VariablesFiltro.java:** Esta clase permite representar las los filtros generados en las consultas del operador filtro de valor.

3. integracion: En este paquete se encuentra la clase que se encarga de agregar los mapeos de integración a las subconsultas generadas. Esta clase también hace uso de la base de datos de "mapeos" donde se encuentra almacenada esta información. La clase que componen este paquete es la siguiente:

- **Integracion.java:** Esta clase se encarga de agregar los mapeos de integración a las consultas generadas. Hace uso de la clase ConexionBD.java para realizar esta función.

4. mediador: En este paquete se encuentra la clase que permite la conexión al modelo de grafos de *Neo4j*. El modelo de grafos está contenido en una carpeta que tiene como nombre "modelo global". Esta se encuentra en la carpeta raíz del proyecto. Además de estos, para procesar la información que retorna el modelo de grafos se necesita agregar una librerías (archivos con extensión .jar) al proyecto. Todas estas librerías se encuentran en la carpeta "librerías" dentro de la carpeta raíz del proyecto. La clase que compone este paquete es la siguiente:

- **Mediacion.java:** Esta clase contiene todo los métodos que permiten acceder a la información del modelo global de Neo4j para la generación de las consultas. En esta clase también

Todos estos paquetes se encuentran en la carpeta "src" dentro de la carpeta raíz del proyecto.

Los resultados de la ejecución de las consultas se almacenan en tres archivos (consultasFuente1.xml, consultasFuente2 y consultasFuente3.xml). Cabe aclarar que estos son para el caso específico de pruebas, el mediador funciona para cualquier número de fuentes y para cualquier modelo global implementado en GDM. Estos tres archivos contienen todas las consultas generadas para las fuentes específicas en la ejecución. Se encuentran en la carpeta "resultados consultas" dentro de la carpeta raíz del proyecto.

Modelo global en *neo4j*

La versión de *Neo4j* utilizada para este proyecto fué "neo4j-community-1.8-windows". Esta versión embebida para java se puede adquirir de la página http://www.neo4j.org/download. Esta contiene todos los .jar necesarios para configurar y acceder a los datos del modelo Para crear el modelo global de Neo4j por primera vez, es decir para generar la carpeta "modelo global", se debe ejecutar una serie de métodos en una clase. La clase que configura el modelo se encuentra en la carpeta src del proyecto por fuera de la paquetería. Esta clase debe ser ejecutada independientemente, y generará la carpeta "Modelo Global." Ya después que haya sido generada esta carpeta se podrá ejecutar el proyecto completo y la clase "Mediacion.java" accederá a la carpeta creada.

Representación de mapeos y mapeos de integración

La versión de *SQLite* utilizada para almacenar los mapeos de las triplas global y los mapeos de integración fue “sqlite 3.7.15”. Para este proyecto solo fue necesaria la inclusión del jdbc al proyecto, ya que se requería solo de la funcionalidad embebida en java. Este archivo se encuentra en la carpeta “librerías” del proyecto y se puede descargar de la página https://bitbucket.org/xerial/sqlite-jdbc/downloads. Para crear la base de datos “mapeos” con *SQLite* se hizo uso de un editor para este tipo de base de datos conocido como “sqliteman”. Este editor permite la crear la base de datos y realizar inserción de datos sobre esta. En el editor creamos la base de datos con el nombre especificado y ejecutamos los scripts para la inserción de los datos y creación de tablas sobre esta. En la carpeta “configuración BD” de la carpeta raíz del proyecto se encuentra el archivo que contiene los scripts para la inserción de los datos y para la creación de las tablas. Estos archivos son “datos.sql” y “tablas.sql” respectivamente. En esta carpeta se encuentra igualmente el editor “sqliteman” dentro de una carpeta llamada “sqliteman 1.2.2”. Este editor es portable. El archivo que debemos ejecutar para que arranque el editor es “sqliteman.exe” que esta contenido dentro de esta última carpeta .Cabe mencionar que cuando se crea una base de datos con sqliteman esta queda por defecto fuera de la carpeta de este, es decir si tenemos la carpeta “sqliteman 1.2.2”. en el escritorio y creamos la base de datos esta quedará en el escritorio.

La siguiente imagen muestra la interfaz del sqliteman, en la parte izquierda se tiene la base de datos cargada y la parte derecha se ejecutan los scripts sobre esta:

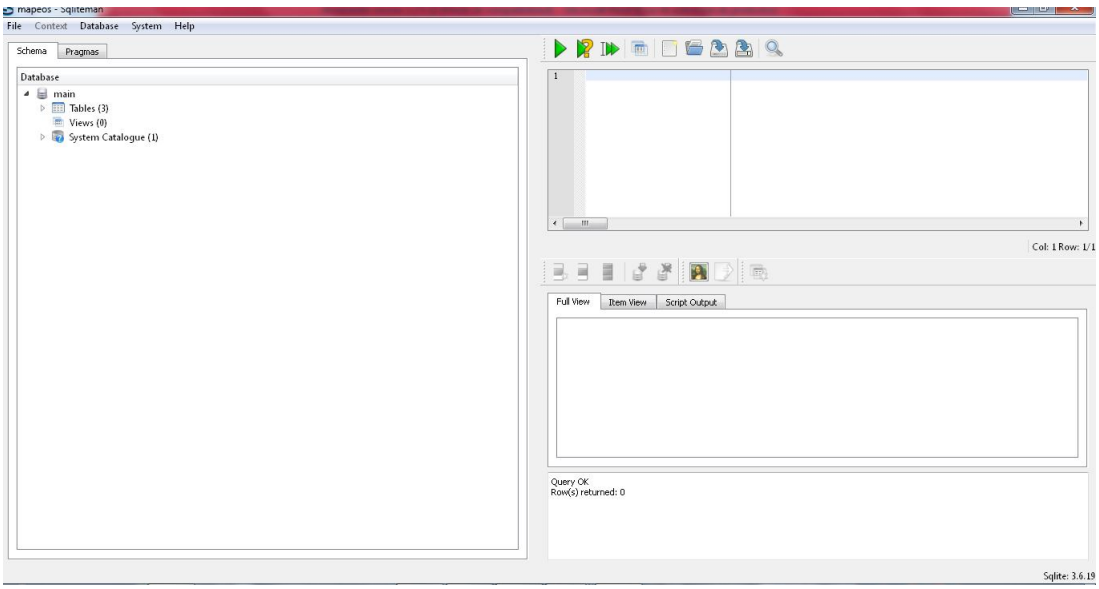


Figura 1. Interfaz sqliteman.

La siguiente imagen muestra la estructura de carpetas del proyecto, donde cada una contiene los archivos especificados anteriormente:

Figura 2. Estructura de Carpetas del Proyecto.

La carpeta bin no fue descrita anteriormente porque no tiene relevancia. Ésta carpeta comúnmente contiene los .class de las clases java del proyecto. En este caso están todos los .class de las clases de los paquetes del proyecto.

Ejecución de una consulta

Con la estructura y configuración detalladas anteriormente, al ejecutar el proyecto se muestra una ventana para navegar entre las carpetas de computador. Esta permite seleccionar el archivo que contiene la consulta que se quiere procesar. Esta consulta dependiendo del tipo de operador debe estar en formato especificado. La siguiente imagen muestra la ventana que se abre al ejecutar la aplicación:

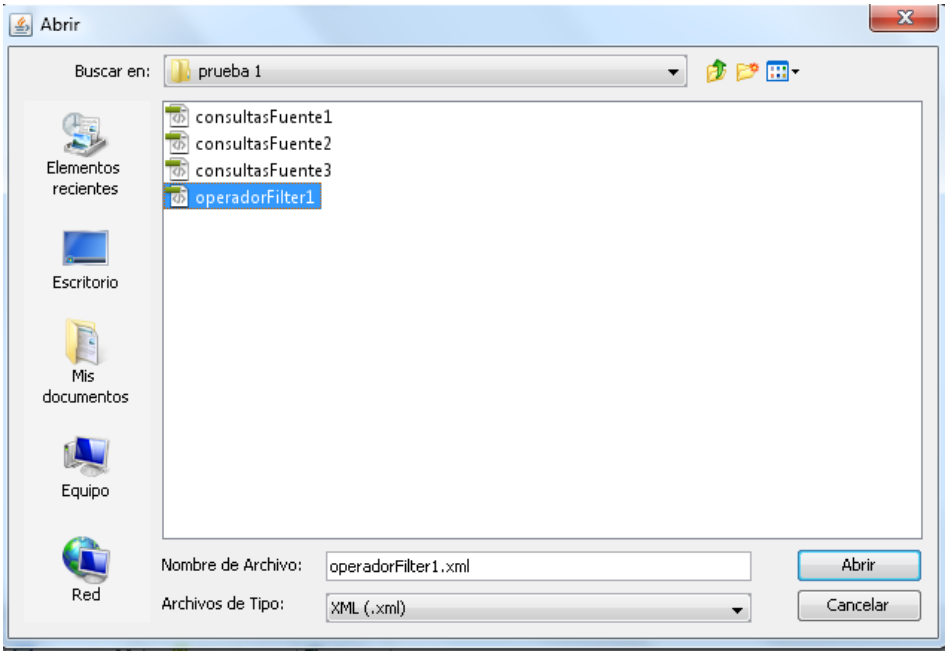


Figura 3. Ventana de Selección de Consultas.

Al seleccionar el archivo que contiene la consulta que se va a procesar, la aplicación genera los tres archivos en la carpeta específica que contiene los resultados para las tres fuentes. En estos archivos se encuentran las consultas definidas en el formato de salida.