Universidad de las Ciencias Informáticas

Facultad 3

Grupo de Investigación de Web Semántica

Método para la integración de datos desde múltiples Bases de Datos Relacionales

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autora:

Rocío Montoya Puentes

Tutor:

Ing. Yusniel Hidalgo Delgado

Ciudad de la Habana, junio de 2014

DECLARACIÓN DE AUTORÍA

Declaro ser autora de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____ _____

Firma del Autor Firma del Tutor

DATOS DE CONTACTO

Síntesis del Tutor

Yusniel Hidalgo Delgado es graduado con título de oro en el 2010 de la carrera de Ingeniería en Ciencias Informáticas por la Universidad de las Ciencias Informáticas (UCI) en La Habana, Cuba. Actualmente investiga sobre nuevos métodos para la publicación y consumo de datos enlazados en varios dominios de aplicación. Es profesor instructor del departamento de técnicas de programación de dicha facultad. Es afiliado de la Sociedad Cubana de Matemática y Computación y de la Asociación Cubana de Reconocimiento de Patrones. Revisor de artículos de la Serie Científica UCI y de la Revista Cubana de Ingeniería de la CUJAE.

A papi, por tanto.

AGRADECIMIENTOS

A mamá y papá por todo

… y por poner por primera vez en mis manos una computadora.

A mi tía Elena, por su incondicionalidad.

A mi primo Ore, por su compañía.

A mi familia, por el apoyo.

A mis amigos, por existir.

A mi tutor, por su paciencia.

A la Universidad de las Ciencias Informáticas, por hacerme crecer.

A todos los que contribuyeron con este trabajo, muchas gracias.

RESUMEN

La Web Semántica es una evolución de la web basada en el manejo de datos semánticos. Sin embargo, la mayoría de los recursos disponibles en la web actual se encuentran contenidos en bases de datos relacionales, las cuales no ofrecen información semántica alguna de los datos. Una de las formas de crear datos semánticos a partir de bases de datos relacionales es transformarlas hacia grafos RDF. Por otra parte, las herramientas existentes para la integración de datos provenientes de múltiples bases de datos relacionales no permiten la generación de una vista integrada dinámica. OpenRefine es una herramienta para el tratamiento de datos que permite la limpieza, transformación y exportación en grafos RDF de los mismos pero que no contempla dentro de sus entradas a las bases de datos relacionales. En este trabajo se describe la obtención de un método para la integración de datos desde de múltiples bases de datos relacionales que, incorporado a OpenRefine en forma de extensión permite la limpieza, transformación y exportación de los datos en grafos RDF, a partir de la generación de una vista integrada dinámica.

Palabras claves: RDF, bases de datos relacionales, integración de datos, OpenRefine, datos enlazados, web semántica

ABSTRACT

The Semantic Web is an evolution of web, which is based in handling of semantic data. However, most of the data sources in the current web are contained in relational databases, which do not have semantic information about data. One way to create semantic data from the databases is to expose it in RDF Schemas. In other hand, current tools to integration data from multiple relational databases don't allow to create an integrated dynamic final view. OpenRefine is a tool for treatment of data, but OpenRefine don't deal with data contained in multiples relational databases. In this job is describes the obtaining of a method to integration of data contained in multiples relational databases that, incorporated to OpenRefine as an extension, allow cleaning, transformation and exportation in RDF Schemas of data through generation of an integrated dynamic view.

Keywords: RDF, relational databases, data integration, OpenRefine, linked data, semantic web

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

La web actual o clásica, presenta un grupo de limitaciones asociadas, en lo fundamental con: el formato de sus documentos y la falta de capacidad de las representaciones en que se basa para expresar significados; la baja integración de las fuentes disponibles, puesto que los datos se encuentran dispersos, sin relación explícita entre ellos, imposibilitando su descubrimiento y utilización por sistemas informáticos; la recuperación imprecisa de información a partir de consultas expresadas en lenguaje natural debido a que la mayoría de los motores de búsqueda actuales están orientados a responder consultas basadas en palabras claves (1).

La web semántica es considerada como la evolución natural de la web clásica con la intención de eliminar las limitaciones antes mencionadas por medio de la adición de metadatos semánticos y ontológicos. Según refiere su propio creador, Tim Berners-Lee, la Web Semántica es una extensión de la web actual, en la que se propicia la cooperación entre los humanos y los agentes inteligentes[1] por medio de la comunicación exitosa de cada uno de ellos con una web más expresiva, en la que la información tiene un significado bien definido (2).

La propuesta de los Datos Enlazados, según Tim Berners-Lee, se refiere a un conjunto de buenas prácticas para la publicación y enlazado de datos estructurados en la Web (3).

El almacenamiento de los datos en la web actual se realiza por medio de bases de datos relacionales. Para obtener una adecuada definición de los datos, la Web Semántica se vale de representaciones como RDF[2]. RDF es una forma de representación de los datos que proporciona información descriptiva sobre los recursos que se encuentran en la web. Es un modelo de datos para los recursos y las relaciones que se pueden establecer entre ellos.

[1] Programas de computadoras que buscan información sin operadores humanos.

[2] Siglas en inglés de Resource Description Framework

Uno de los componentes tecnológicos que falla en estos procesos es el que permite integrar datos provenientes desde múltiples fuentes para su posterior transformación y conversión a RDF. Estudios relacionados con el tema proponen soluciones que consisten en la transformación de las bases de datos relacionales en esquemas RDF haciendo uso de sistemas para la integración de datos, como RDB2RDF. Estos sistemas están basados en la filosofía para la transformación de bases de datos relacionales en grafos RDF adoptada por el RDB2RDF Working Group (WG).[3]

El caso particular de transformar múltiples bases de datos relacionales a esquemas RDF también ha sido tratado y su solución se basa en el uso de los sistemas MRDB2RDF (4). Los retos que enfrenta este último tipo de sistemas están asociados a la baja flexibilidad de la vista que se genera a partir de la integración de los datos, puesto que los mismos, una vez se encuentren en esta estructura (ontologías, en el caso particular de RDF2RDB y MRDB2RDF), no pueden ser editados antes de ser transformados en grafos RDF.

OpenRefine es una herramienta de código libre que, a partir del uso de la extensión GRefine, permite la transformación y exportación en formato RDF de un conjunto de datos. Esta herramienta tiene dentro de sus formatos de entrada permitidos TSV, CSV, *SV, Excel (.xls y .xlsx), JSON, XML, RDF como XML y documentos de Google Data, pero no existe la posibilidad de importar datos directamente a partir de bases de datos relacionales ni de obtener una vista integrada dinámica de datos provenientes de múltiples bases de datos relacionales.

Atendiendo a lo anteriormente explicado se plantea como problema a resolver:

¿Cómo integrar datos desde múltiples bases de datos relacionales en la herramienta OpenRefine para la posterior transformación y exportación de los mismos en grafos RDF?

[3] http://www.w3.org/2001/sw/rdb2rdf

La investigación tiene como objeto de estudio: “Métodos para la integración de datos” y se enmarca específicamente en el campo de acción: “Métodos para la integración de datos desde de múltiples bases de datos relacionales”.

Para dar solución al problema se plantea como objetivo general:

- Obtener un método para la integración de datos desde múltiples bases de datos relacionales que, incorporado a OpenRefine en forma de extensión, permita la transformación y exportación de los datos en grafos RDF.

Para dar cumplimiento al objetivo general, se definieron los siguientes objetivos específicos:

1. Definir el marco teórico y el estado del arte alrededor de los principales enfoques de integración de datos mediante la consulta de literatura científica actualizada para identificar tendencias y adoptar una posición al respecto.
2. Obtener un método para la integración de datos desde múltiples bases de datos relacionales a partir de la identificación de los elementos que intervienen en el proceso.
3. Obtener los artefactos ingenieriles correspondientes a la fase de diseño de la propuesta de solución para su posterior implementación.
4. Implementar el método propuesto a través del desarrollo de una extensión de OpenRefine que posibilite la posterior transformación y exportación de en grafos RDF de la vista dinámica obtenida.
5. Validar que el método propuesto y la extensión implementada resuelven el problema de la integración de datos provenientes de múltiples bases de datos relacionales.

Atendiendo al problema de la investigación se plantea como idea a defender:

- Si se integran los datos desde múltiples bases de datos relacionales en OpenRefine, entonces será posible la posterior transformación y exportación en grafos RDF de los mismos.

Teniendo como variable independiente: Método para la integración de los datos provenientes de múltiples bases de datos relaciones; y como variable dependiente: transformación y exportación en grafos RDF de los datos.

Como métodos teóricos se utilizaron:

- Método analítico-sintético: Para analizar y distinguir los elementos que componen los procesos de integración de datos provenientes de múltiples recursos. El método permitirá el estudio de cada uno de estos elementos por separado y su posterior integración como una nueva unidad, partiendo de la comprensión total de la esencia de los mismos. Además, el método se empleará para examinar las características positivas y negativas de los enfoques existentes para la integración de datos.
- Inductivo-deductivo: Para la organización y análisis del contenido a partir de lo más general hasta lo más específico.
- Análisis histórico-lógico: Para realizar un estudio de los antecedentes y conceptos asociados a los procesos de integración de datos provenientes de múltiples recursos.

Como métodos empíricos se utilizaron:

- Observación: Para la observación de los resultados del uso de la solución propuesta.
- Experimento: Para la validación de la solución a partir de casos de estudio.
- Medición: Para la evaluación de los resultados de la aplicación de pruebas a la solución.

Para garantizar el cumplimiento de los objetivos propuestos para la investigación, el documento se estructuró de siguiente manera:

Capítulo 1: Se realizó una breve introducción al objeto de estudio y su novedad científica, impacto social y relevancia. Se expusieron los resultados obtenidos a partir de un análisis bibliométrico y documental de parte de la literatura científica referente al tema de la integración de datos. Se definió el marco teórico de la investigación a partir de la definición de los conceptos fundamentales asociadas a

la temática de la misma. Además, se realizó un estudio de las principales tendencias o enfoques de integración.

Capítulo 2: Se realizó una descripción de la propuesta y se ejecutó cada una de las fases propuestas por la metodología de desarrollo utilizada.

Capítulo 3: Se realizaron pruebas a la propuesta de solución obtenida y finalmente se validó la misma a partir de un caso de estudio.

CAPÍTULO I: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

La publicación de los datos estructurados en la web es, en sí misma, insuficiente. La información se encuentra desorganizada y, en muchos casos, redundante. Los usuarios de la web podrían hacer un mejor aprovechamiento de los datos si estos estuvieran relacionados con otros datos publicados.

Para posibilitar tal relación entre los datos y, sobre todo, para facilitar la comprensión de los mismos, surge el concepto de web semántica. Según palabras de Tim Berners-Lee, su creador y precursor, “La Web Semántica no pretende sustituir la Web actual, sino que es una extensión de la misma en la que la información tiene un significado bien definido, posibilitando a los humanos y las computadoras trabajar en cooperación.” (2)

Para lograr una adecuada definición de los datos, la web semántica se vale de tecnologías como RDF, la cual se define como un modelo de datos para los recursos y las relaciones que se pueden establecer entre ellos. A partir de la migración a la que debe ser sujeta la información actual de la web, la cual se almacena mayoritariamente en bases de datos relacionales, emergen la integración de datos y la generación de grafos RDF como problemáticas para la comunidad de investigadores sobre la web semántica. De esta manera, surgen algunas preguntas científicas como:

1. ¿Cuáles son las aproximaciones básicas para realizar la alineación de las bases de datos relacionales al esquema global?
2. ¿Cuáles son los principales enfoques para la Integración de datos?
3. ¿Qué tecnología soporta el proceso de integración de datos?

En este capítulo se presentan los fundamentos básicos del problema de la integración de datos así como los conceptos y definiciones fundamentales. Además, se definen las herramientas, tecnologías, marcos de trabajo y otros componentes utilizados en el desarrollo de la propuesta de solución.

1.2. Marco teórico. Conceptos y definiciones

La Web Semántica surge como una evolución de la web actual encaminada a dar solución a sus principales limitaciones por medio de la aplicación de un conjunto de buenas prácticas para representar y compartir datos. De esta manera, es posible describir recursos, establecer relaciones de significado entre ellos y descubrir nuevo conocimiento implícito.

Con la finalidad de lograr la transición de la web actual hacia la web semántica, Tim Berners-Lee presenta el concepto de los datos enlazados. De acuerdo a su definición, se llega a la conclusión la idea básica de los datos enlazados es aplicar la arquitectura general de la web a la tarea de compartir datos estructurados a escala global (3).

Para la definición de los datos, la web clásica utiliza las bases de datos relacionales. Las bases de datos relacionales, introducidas por Codd en 1970 (5), son un tipo de sistema de bases de datos que utiliza relaciones (tablas) para la organización de los datos. No obstante a sus potencialidades para el almacenamiento de datos, las bases de datos relacionales carecen de semántica, puesto que las tablas y las relaciones que contienen no brindan una información descriptiva de su contenido.

Para lograr una correcta publicación de datos en el ámbito de la web semántica se requiere una alta compatibilidad y estandarización para la representación de los mismos, siguiendo cada uno de los principios de los datos enlazados, los cuales son (3):

1. Use URIs[4] como nombres para las cosas.

2. Use URIs HTTP, de modo que las personas puedan ver esos nombres.

3. Cuando alguien ve una URI, proporciona información útil, usando los estándares (RDF, SQPRQL).

4 URI, siglas en inglés de Uniform Resource Identifier. Es una cadena de caracteres corta que identifica inequívocamente un recurso

4. Incluye enlaces a otras URIs, de modo que ellos puedan descubrir más cosas.

Para lograr el cumplimiento de los principios de los datos enlazados, es necesario adoptar estándares establecidos por la W3C[5] como RDF.

RDF es un modelo de datos flexible basado en grafos, útil para describir datos estructurados y sus interrelaciones en un formato procesable por las computadoras. Se basa en tripletas de la forma sujeto-predicado-objeto (6). Al conjunto de tripletas RDF relacionadas se le denomina grafo RDF[6].

De acuerdo al propio concepto de datos enlazados, puede entenderse que los mismos tienen como finalidades propiciar una adecuada estructuración del formato para los recursos web y garantizar la integración de datos provenientes de fuentes de datos heterogéneas que pueden ser mantenidas por organizaciones con diferentes localizaciones geográficas.

En (4) se define la integración de datos como un conjunto estructurado de procesos encaminados a la obtención de una vista centralizada y única a partir de datos provenientes de múltiples fuentes de datos. En tanto, Katsis y Papakonstantinou son más simples en (7) al definirla como el problema de encontrar y combinar datos de diferentes fuentes autónomas y, generalmente, heterogéneas. Otras definiciones se proponen en (8, 9).

De manera general, todas las conceptualizaciones son acertadas y, al combinar estas perspectivas, puede conceptualizarse la integración de datos como:

Definición 1: La integración de datos es el conjunto estructurado de procesos encaminados a la obtención de una vista centralizada y única a partir del descubrimiento y combinación de los datos provenientes de múltiples recursos.

[5] http://www.w3c.org

[6] http://www.w3.org/TR/rdf-concepts/#section-rdf-graph

1.3. Sistemas de Integración de Datos

1.3.1. Definición de Sistema de Integración de Datos

Enfocados a resolver los principales inconvenientes a los que se enfrenta la integración de datos surgen los Sistemas de Integración de Datos (SID). Un sistema de integración de datos es una formalización teórica para el problema de la integración de datos que involucra un esquema global, un conjunto de fuentes de datos (esquema de recursos) y un conjunto de declaraciones que especifican cómo se realiza el mapeo entre los dos esquemas implicados. Además, un sistema de integración de datos provee una interfaz uniforme para multitud de recursos, de manera que libera al usuario de la realización de extensas búsquedas a través de recursos aislados y le permite concentrarse en qué buscar en lugar de cómo hacerlo.

1.3.2. Arquitectura de los Sistemas de Integración de Datos

Aun cuando existen algunos prototipos para el diseño de un sistema de integración de datos, en cada de unos pueden identificarse elementos en común. De esta forma es posible referirse a una arquitectura (7).

De acuerdo a su arquitectura, el sistema de integración de datos se comunica con el esquema de recursos haciendo uso de los “wrappers”. Un wrapper (10) convierte los datos provenientes de un recurso en específico a un formato entendible por el procesador de consultas del sistema de integración de datos. Otra de las características de esta arquitectura es que las consultas no se realizan directamente al recurso donde los datos están almacenados sino que se realizan al esquema global y luego estas consultas son reformuladas, optimizadas y distribuidas al recurso o los recursos que contengan los datos involucrados en la consulta (8).

1.3.3. Clasificación de los Sistemas de Integración de Datos

En (7) y (11) se proponen distintas clasificaciones para los sistemas de integración de datos. Estas son:

- Sistemas de Integración de Datos basados en Vistas (VDISs, por sus siglas en inglés): proveen unos puntos únicos de acceso para los recursos integrados. Cuando se genera una única vista se trata de un Sistema Centralizado de Integración de Datos (SCID).
- Sistemas de Integración de Datos basados en Ontologías: En un sistema de integración de datos basado en ontologías, el esquema global es representado como una ontología simple, como múltiples ontologías o como una combinación de ambas en la que una gran ontología es usada como una vista para múltiples ontologías pequeñas. Los recursos pueden estar representados por varios tipos de datos, entre ellos bases de datos relacionales. El esquema típico de un sistema de integración de datos basado en ontologías involucra dos componentes esenciales: un mediador y los "wrappers". Un "wrapper" es usado para interactuar directamente con el recurso y obtener los resultados de las consultas que hacia él se realiza. El mediador actúa como distribuidor de las consultas a cada uno de los "wrappers" y como mecanismo de integración de los resultados de las mismas (12).

1.4. Enfoques de integración de datos

Existen diferentes maneras de enfocar la integración (13). Las principales son:

I. Replicación de Datos (13, 14)
II. EAI (14–17)
III. EII (10, 14, 18)
IV. ETL [7] (14, 19–21)

1.4.1. Replicación de datos

La replicación de datos consiste en la comunicación o transporte de datos entre dos o más servidores con el objetivo de aumentar la disponibilidad de los datos y mejorar

[7] Del inglés Extract, Transform and Load

el rendimiento de las consultas globales al permitir que ciertos datos de la base de datos estén almacenados en más de un sitio.

Integración de Aplicaciones Empresariales (EAI)

La Integración de Aplicaciones de Empresariales (EAI en Inglés) se define como el uso de software y principios de arquitectura de sistemas para integrar un conjunto de aplicaciones.
EAI se utiliza para la integración de Aplicaciones – a– Aplicaciones y está orientado a mensajes, no a la integración de datos en lotes y abundantes transformaciones.

1.4.2. Integración de Información Empresarial (EII)

La Integración de Información Empresarial (EII en inglés) es un mecanismo de transformación y acceso a datos que provee una vista unificada de múltiples recursos de información de la organización, que incluyen, entre otros, bases de datos y aplicaciones. La principal característica radica en que no se basa en una integración Base de Dato-Base de Dato, sino que el mecanismo se basa en la generación de una interfaz para la recuperación de los datos sin hacer una copia o replicación de los mismos en algún espacio físico de almacenamiento. Usualmente este método resulta en un sistema de información heterogéneo distribuido, virtualmente Integrado.

Extracción, Transformación y Carga de Datos (ETL)

ETL (*Extract, Transform and Load*) es la tecnología enfocada a la integración de datos, tanto por lote, como a tiempo real hacia almacenes de datos.

ETL permite la consolidación de datos para la construcción de bases de datos permanentes que sean de utilidad tanto para tareas de consulta, como de análisis,

creación de dashboards[8], transformación y limpieza, reformateo y migración de datos, creación de repositorios, datamarts[9] y la comunicación entre aplicaciones.

A continuación se muestra una tabla comparativa a partir de aspectos técnicos asociados a cada uno de los enfoques de integración de datos mencionados. (Ver Tabla I).

1.4.3. Comparación técnica entre EII, EAI y ETL

Tabla I: Comparación entre ETL, EII y EAI a partir de sus características técnicas

	ETL	EII	EAI
Flujo de datos	Unidireccional. Del recurso al esquema global.	Bidireccional	Bidireccional
	Peso = 1	Peso = 2	Peso = 2
Complejidad de las transformaciones	Alta	Media	Baja
	Peso = 3	Peso = 2	Peso = 1
Tamaño de los volúmenes de datos procesables	Grande	Pequeño	Mediano
	Peso = 3	Peso = 2	Peso = 1
Versionado y soporte	Continuado	Limitado	Limitado
	Peso = 2	Peso = 1	Peso = 1

Nota 1: La Replicación de datos no figuró dentro de la comparación puesto que se considera una forma simplificada de ETL.

[8] Referente a Tablero de Control Digital. Es la aplicación digital del Cuadro de Mando Integral. Brinda una visión general del comportamiento de la institución.

[9] Versión especial de almacén de datos. Son subconjuntos de datos con el propósito de ayudar a que un área específica dentro del negocio pueda tomar mejores decisiones.

Nota 2: Durante la comparación se asignó un peso que cuantifica los valores cualitativos de las características de los tres enfoques relacionados. Tal peso estuvo determinado por: (3 – Excelente, 2 - Regular, 1 - Deficiente).

Para lograr una mejor comprensión de los resultados se elaboró un gráfico que utiliza las ponderaciones otorgadas en la comparación anterior (ver Figura 32).

Dentro de los principales enfoques de integración, ETL resalta como el más robusto, completo y aplicable a múltiples dominios. Engloba de forma general los beneficios de las otras alternativas en tanto que, sus principales inconvenientes, asociados a la gran cantidad de operaciones que realiza y que involucra a todo el volumen de datos, se ven compensados en alguna medida por la calidad de los resultados obtenidos.

1.5. Perspectiva teórica de la integración de datos

1.5.1. Conceptualización teórica

Una formalización teórica para el problema de la integración de datos involucra tres componentes principales: el esquema global, el esquema de recursos (fuentes de datos) y el mapeo de dichos recursos hacia el esquema global, constituido por un conjunto de aserciones. Las fuentes de datos contienen los elementos atómicos de información reales. Por otra parte, tanto el esquema global se define como el nivel que representa a los datos integrados y que provee los elementos necesarios para expresar las consultas sobre el sistema de integración de datos. Las aserciones en el mapeo contienen las conexiones entre los elementos del esquema global y los de los recursos.

Sea I un sistema de integración de datos (11, 12, 22). Su formalización en términos de una tripleta se expresa como (G, S, M), donde:

- G es el esquema global, expresado en un lenguaje L_G sobre un alfabeto A_G. El alfabeto comprende un símbolo para cada elemento de G (por ejemplo, relación si G es relacional, clase si G es orientada a objetos, etc.)

- S es el esquema de recursos, expresado en un lenguaje L_S sobre un alfabeto A_S. El alfabeto incluye un símbolo por cada elemento de los recursos.
- M es el mapeo entre G y S, constituido por un conjunto de aserciones de la forma:

$$q_s \rightarrow q_g$$

$$q_g \rightarrow q_s$$

Donde q_s y q_g son dos consultas de la misma aridad, respectivamente sobre el esquema de recursos S y el esquema global G. Las consultas q_s están expresadas en un lenguaje de consultas $L_{M,S}$ sobre el alfabeto A_S y las consultas q_g están expresadas en un lenguaje de consultas $L_{M,G}$ sobre el alfabeto A_G. Una aserción $q_s \rightarrow q_g$ específica que el concepto representado por la consulta q_s sobre los recursos correspondes al concepto en el esquema global representado por la consulta q_g. (De forma similar ocurre para una aserción del tipo $q_g \rightarrow q_s$).

Las consultas al sistema de integración de datos I están dirigidas al esquema global G y se expresan en el lenguaje L_G sobre el alfabeto A_G. Una consulta está encaminada a proveer especificación sobre qué datos extraer de la base de datos virtual representada por el sistema de integración de datos.

La anterior definición teórica de sistema de integración es una generalización. La naturaleza de cada aproximación para un sistema de integración depende de las características del mapeo, del poder expresivo de los datos contenidos en los recursos y de la complejidad de los lenguajes de consulta.

Uno de los aspectos más importantes en el diseño de un sistema de integración de datos constituye la especificación de las aserciones del mapeo. La literatura referente al tema (8, 11, 12) propone dos aproximaciones básicas:

- Local_as_view
- Global_as_view

1.5.2. Local_As_View

En un sistema de integración de datos basado en la aproximación LAV, el mapeo M asocia a cada elemento s del esquema de recursos S una consulta q_g sobre el esquema global G. El lenguaje de consultas $L_{M,S}$ solamente permite aserciones constituidas por un símbolo del alfabeto A_S. De esta manera se entiende que el mapeo en LAV está constituido por un conjunto de aserciones (una por cada elemento s de S) de la forma $s \rightarrow q_g$.

En (7) la forma de proceder en LAV se define básicamente a partir de una función que hace corresponder a cada recurso su respectiva vista en el esquema global. Una representación sería:

$$I(R_i) \rightarrow U_i$$

Donde U_i es una consulta sobre el esquema global e I representa al SID como la función de correspondencia entre cada recurso R_i y U_i.

Desde el punto de vista de la modelación, el mapeo LAV está basado en la idea de que cada elemento s de S debe estar caracterizado en términos de una vista q_g sobre el esquema global G. De esta manera se entiende que LAV tiene un enfoque declarativo, puesto que especifica qué información del esquema global está contenida en cada uno de los recursos.

La propuesta de LAV es efectiva siempre que el sistema de integración de datos esté basado en un esquema global G estable. Esta aproximación favorece la extensibilidad del sistema. La adición de una nueva fuente de datos implica únicamente la creación de una aserción, puesto que el esquema global es establecido independiente de los recursos y estos, a su vez, son definidos como vistas sobre el esquema global.

Las mayores dificultades de LAV radican en la imposibilidad de modelar recursos que contienen información no especificada en el esquema global y en la complejidad del mecanismo de procesamiento de consultas, puesto que cada vista sólo provee información parcial de los recursos.

1.5.3. Global_As_View

En un sistema de integración de datos basado en la aproximación GAV, el mapeo M asocia a cada elemento g del esquema global G una consulta q_s sobre el esquema de recursos S. El lenguaje de consultas $L_{M,G}$ solamente permite aserciones constituidas por un símbolo del alfabeto A_G. De esta manera se entiende que el mapeo en GAV está constituido por un conjunto de aserciones (una por cada elemento g de G) de la forma $g \rightarrow q_s$.

En (7) la forma de proceder en GAV se define básicamente a partir de una función que devuelve como una vista el resultado de una consulta sobre el esquema de recursos. Una representación sería:

$$V_i \rightarrow I(R_i)$$

Donde V_i representa la vista asociada al recurso R_i e I representa al SID como una función matemática.

Desde el punto de vista de la modelación, el mapeo GAV está basado en la idea de que cada elemento g de G debe estar caracterizado en términos de una vista q_s sobre el esquema de recursos S. De esta manera se entiende que GAV tiene un enfoque procedural, puesto que describe cómo se genera el esquema global a partir de consultas a cada uno de los recursos. Dicho de otra forma, si se requiere una evaluación de los datos en el esquema global (o sea, si se realiza una consulta a G), la aproximación GAV permite al sistema recuperar los datos trasladando las respectivas consultas al esquema de recursos.

La literatura referente a GAV comúnmente se refiere a las vistas en S asociadas a cada $g \in G$ como "*sound*", debido a que todos los datos proveídos por una vista satisfacen el correspondiente elemento en el esquema global, pero puede existir información adicional que también lo satisface y que no fue proveída por la vista.

La propuesta de GAV es efectiva siempre que el sistema de integración de datos esté basado en un esquema de recursos S estable. Esta concepción destaca por la

simplicidad a la hora de definir las aserciones y su fortaleza como procesador de consultas.

Por GAV, la extensión del sistema con un nuevo recurso puede ser compleja puesto que este tiene un impacto en la definición de varios elementos en el esquema global, cuyas vistan asociadas tienen que ser redefinidas.

Debido a que el esquema global se construye directamente a partir del esquema de recursos, no puede crearse en G ninguna relación que no esté previamente definida en las fuentes y tiene que estar explícitamente especificada la forma en que los datos van a ser combinados.

1.5.4. Comparación entre LAV y GAV

Para LAV, el procesamiento de consultas presenta dificultades, puesto que el único conocimiento que se tiene de los datos es una vista en el esquema global que los representa y no se accede realmente a ellos. En contraste, GAV convierte al sistema de integración de datos en un potente procesador de consultas, puesto que existe una recuperación directa de los datos en el esquema de recursos (12).

LAV permite la incorporación de nuevos recursos con una relativa facilidad, en tanto para GAV esta es una tarea más compleja.

Desde el punto de vista de la modelación, mientras que en LAV, el diseñador del sistema ha de concentrar los esfuerzos en la especificación del contenido de los recursos en términos del esquema global, en GAV es necesario especificar concretamente cómo obtener los datos del esquema global a partir de consultas directamente sobre el esquema de recursos.

1.5.5. Álgebra Relacional

El Álgebra Relacional puede ser utilizada para operaciones sobre las bases de datos relacionales y constituye la base teórica para el proceso de integración de datos. Contiene una serie de operadores que pueden ser aplicados sobre tuplas

relacionales. Una tupla relacional es un conjunto de atributos-valores. Algunos de los operadores más utilizados en el álgebra relacional (4) son:

Sean A, B y C conjuntos, R una tupla relacional, $a_1, \dots, a_n$ atributos que pertenecen a R:

- Unión ($\cup$): El resultado de $A \cup B$ es el conjunto C que contiene (sin duplicaciones) todos los elementos de A y B.
- Diferencia (\): El resultado de $A \backslash B$ es el conjunto C que contiene todos los elementos de A que no se encuentran en B.
- Intersección ($\cap$): El resultado de $A \cap B$ es el conjunto C que contiene todos los elementos que existen tanto en A como en B.
- Selección (σ): Una selección σ sobre un conjunto de tuplas relacionales retorna un subconjunto de R en el cual se cumple que σ es verdadera.
- "Join" natural: A Join B retorna un conjunto que contiene elementos de A y B que comparten un mismo atributo-valor.
- "Join" cruzado ($\times$): $A \times B$ retorna un conjunto que contiene todos los atributos de A y B.
- Proyección (π): Una proyección $\pi_{a_1,\dots,a_n}(R)$ es una selección de atributos $a_1, \dots, a_n$ sobre un conjunto de tuplas relacionales R.

1.6. Procesador de Consultas Distribuidas

El Procesador de Consultas Distribuidas[10] (4), constituye la principal tecnología en un Sistema de Integración de Datos. Un DQP toma una consulta global, la particiona y la distribuye como consultas locales a cada una de las bases de datos locales e integra el resultado de cada una en un resultado global.

A partir de lo anteriormente explicado, puede entenderse que existen dos tipos de consultas:

- Consulta SQL global: involucra a múltiples bases de datos relacionales.

[10] DQP, siglas en inglés Distribuited Query Processor

- Consulta SQL local: involucra solamente a una base de datos relacional.

Existen varias implementaciones para un DQP. Dos de ellas son OGSA-DQP y UnityJDBC, las cuales serán analizadas a continuación.

1.6.1.1. OGSA-DQP

OGSADAI es un sistema de integración de datos que tiene una naturaleza distribuida y una arquitectura orientada a servicios. OGSA-DQP es la implementación de un DQP diseñado especialmente para trabajar como mediador en OGSADAI, en un nivel superior a los “wrappers”. OGSA-DQP posee dos componentes principales, los cuales son el Coordinator y el Evaluator. El Coordinator se encarga de la optimización y distribución de la consulta global, mientras que el Evaluator se encarga de la ejecución y evaluación de las consultas locales (23).

1.6.1.2. UnityJDBC

Un manejador Java Database Connectivity (JDBC) (24) es un módulo que actúa como interfaz para un sistema de múltiples bases de datos relacionales. UnityJDBC es una extensión del estándar JDBC, adaptada al trabajo con múltiples bases de datos relacionales y funciona a través de una arquitectura cliente-servidor.

1.7. Tecnologías utilizadas para el desarrollo

1.7.1. Lenguaje de modelado

El lenguaje de modelado se puede conceptualizar como una estandarización de notaciones y reglas que permitan diseñar gráficamente un sistema o parte de él. Para este propósito se seleccionó el Lenguaje Unificado de Modelado (UML, por sus siglas en inglés).

UML, es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. Ofrece un estándar para describir un plano del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio, funciones del

sistema y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables (25).

1.7.2. Herramienta CASE

Las herramientas de Ingeniería de Software Asistida por Computadora (Computer Aided Software Engineering (CASE)), se pueden definir como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores durante todos los pasos del ciclo de vida de desarrollo del software. [11]

Para asistir el proceso de desarrollo de la propuesta de solución se utilizó Visual Paradigm for UML 8.0.

1.7.3. Lenguajes de programación

Java:

Los lenguajes de programación son tecnologías que permiten crear soluciones de software. Existe un gran número de lenguajes que siguen la idea de lograr la mayor abstracción posible y facilitar el trabajo al desarrollador, aumentando la productividad en función del resultado alcanzado. (26)

Dado que la propuesta de solución se implementa a partir de una extensión para OpenRefine, el lenguaje de programación del lado del servidor utilizado para la codificación de la misma debe ser altamente compatible con el utilizado por la herramienta. De esta manera, Java resulta el más apropiado para la implementación de la propuesta de solución.

Del paquete para el desarrollo en Java [12] se utilizó la versión 1.7.

Javascript 1.8:

[11] http://case-tools.org/

[12] Más conocido por sus siglas en inglés JDK, proveniente de Java Development Kit, es un software que provee herramientas de desarrollo para la creación de programas en Java.

Es un lenguaje de programación interpretado, por lo que no requiere compilación. Es utilizado en el lado del cliente, permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. JavaScript es orientado a objetos, imperativo y dinámico. (27)

1.7.4. Marcos de trabajo y bibliotecas

Un marco de trabajo (framework, según su denominación en idioma inglés) es una estructura conceptual y tecnológica de soporte definido, que propone una arquitectura de archivos y utilidades que aceleran la programación de una aplicación informática, teniendo una metodología de trabajo que facilita la generación de aplicaciones, módulos de uso común y formularios que permiten al desarrollador centrarse principalmente en los aspectos específicos de la aplicación. (28)

Butterfly:

OpenRefine hace uso del marco de trabajo Butterfly para proveer una arquitectura para sus extensiones. A partir de esto, para el desarrollo de la propuesta de solución fue utilizada la arquitectura de archivos y subdirectorios que propone Butterfly.

Velocity 1.5:

Velocity es un gestor de plantillas basado en Java que facilita la incorporación de contenido dinámico en las páginas web. [13] Como parte del desarrollo de la propuesta de solución y, en consonancia con la implementación de OpenRefine, se utilizó marco de trabajo Velocity 1.5.

jQuery:

Para simplificar el proceder con los documentos HTML, se utilizó la biblioteca jQuery 1.9.1, con facilidades para la manipulación del Modelo de Objetos del Documento (DOM, por sus siglas en inglés), de eventos, funcionalidades para desarrollar animaciones y agregar interacción a páginas web utilizando AJAX.

13 https://velocity.apache.org

Para añadir efectos visuales a las páginas web, se utilizó la biblioteca jQueryUI 1.10.3.

1.7.5. Herramientas de desarrollo

Para el desarrollo del componente se seleccionó un conjunto de herramientas las cuales se exponen a continuación:

NetBeans 7.4:

El código de OpenRefine contiene un proyecto de Eclipse, el cual puede ser importado hacia NetBeans. Eclipse fue desarrollado como un IDE[14] multipropósito, de acuerdo a la definición que da el proyecto Eclipse acerca de su software, la cual planeta que el mismo es: "una especie de herramienta universal - un IDE abierto y extensible para todo y nada en particular".[15] Por otra parte, NetBeans IDE fue desarrollado especialmente para el trabajo en Java[16], lo que constituye aproximadamente el 80% del total del trabajo de implementación. Por tal motivo, fue seleccionado NetBeans para el desarrollo de la propuesta de solución.

PostgreSQL 8.3.4:

PostgreSQL es un Sistema Gestor de Bases de Datos Objeto-Relacional (ORDBMS, según sus siglas en inglés) libre.

Características principales de PostgreSQL utilizadas en el desarrollo de la propuesta de solución: [17]

- Aislamiento: Asegura que una operación no puede afectar a otras. Esto asegura que dos transacciones sobre la misma información nunca generará ningún tipo de error.

[14] Siglas en inglés de Integrated Development Enviroment

[15] http://www.eclipse.org

[16] http://netbeans.org

[17] http://www.postgressql.org

- Durabilidad: Asegura que una vez realizada la operación, esta persistirá y no se podrá deshacer.
- Soporte de casi todos los sistemas operativos: Linux, Unix, Mac OS, Beos, Windows, entre otros.
- Documentación: Posee una vasta documentación bien organizada, pública y libre.

Apache Ant 1.9.2:

Apache Ant es una herramienta utilizada en programación para la realización de tareas mecánicas y repetitivas, normalmente durante la fase de compilación y construcción (build). Es, por tanto, un software para procesos de automatización de compilación. Apache Ant está desarrollado en Java, por lo que resulta más apropiado para la construcción de proyectos en dicho lenguaje de programación. (29) Para el proceso de compilación del código de la implementación de la propuesta de solución se utilizó Apache Ant en la versión 1.9.2.

Firebug 1.12.4:

Firebug es una extensión para Firefox que proporciona gran cantidad de herramientas de desarrollo web a su alcance mientras se navega. Se puede editar, depurar, perfilar y monitorear CSS, HTML, JavaScript y HttpRequests en cualquier página web en tiempo real. [18]

Para el desarrollo de la propuesta de solución fue utilizado Firebug en su versión 1.12.4 para llevar un control del tráfico de información desde el cliente hacia el servidor y viceversa.

1.8. Metodología de desarrollo

El desarrollo de todo software debe estar guiado por una metodología de desarrollo. La metodología permite estructurar un conjunto de actividades de acuerdo a un orden lógico. Para dar cumplimiento a cada una de estas actividades se hace uso

[18] http://getfirebug.com

de herramientas, técnicas y modelos que influyen directamente en la calidad del producto final. Existen dos grupos de metodologías para el desarrollo de software: las tradicionales y las ágiles.

Las metodologías tradicionales centran su atención en llevar una completa documentación, planificación y procesos de todo el proyecto. Por otra parte, las metodologías ágiles están orientadas a entornos variables, proyectos pequeños donde los individuos y las interacciones entre ellos son más importantes que las herramientas y los procesos empleados y, en donde se exige reducir drásticamente los tiempos de desarrollo manteniendo una alta calidad.

La prioridad del enfoque ágil radica en satisfacer al cliente mediante tempranas y continuas entregas que un determinado valor del negocio, por lo que es más importante crear un producto de software que funcione sin tener que escribir documentación exhaustiva, tributando con una elevada simplificación pero sin renunciar a las prácticas esenciales para asegurar la calidad del producto. (30)

Dada la necesidad de desarrollar la propuesta de solución en un breve período de tiempo, garantizando además la flexibilidad necesaria en cuanto a la variación de los requisitos y el manejo de los riesgos técnicos, así como reducir la generación de documentos y artefactos, se hace necesario optar por un enfoque ágil de desarrollo de software en lugar de un enfoque tradicional o pesado.

Teniendo en cuenta lo anterior se evaluaron varias metodologías que siguen el enfoque ágil de desarrollo de software. Tal es caso de Scrum, Crystal y Extreme Programming (XP).

Teniendo en cuenta que la dimensión del proyecto es pequeña, que el mismo está basado en el uso de nuevas tecnologías que suman los riesgos técnicos asociados a ellas, que Extreme Programming (XP) se ajusta eficientemente a proyectos con estas características y que su sencillez permite que los equipos jóvenes pueden incorporarla de manera más natural, se destaca la idoneidad de esta metodología para el desarrollo de la solución.

El ciclo de vida ideal de XP se compone de seis fases: Exploración, Planificación de la Entrega (Release), Iteraciones (de no más de tres semanas), Producción, Mantenimiento y Muerte del Proyecto.

1.9. Conclusiones del capítulo

La revisión bibliográfica de la literatura científica permitió conocer que la replicación de datos, la integración de aplicaciones empresariales, la integración de información empresarial y la extracción, transformación y carga son los principales enfoques para la integración de datos.

El estudio de las bases teóricas para los sistemas de integración de datos arrojó que un sistema de integración de datos cuenta con tres elementos fundamentales: un esquema de recursos, un esquema global y un conjunto de aserciones que especifican cómo se realiza la alineación entre ambos esquemas implicados.

Además, el entendimiento de los principales elementos del Álgebra Relacional permitió concluir en que el proceso de integración de datos tiene su base matemática en un problema de intersección algebraica.

Como parte del estudio bibliográfico, también se puedo llegar a la conclusión de que el procesador de consultas distribuidas constituye la principal tecnología en un sistema de integración de datos.

CAPÍTULO II. DISEÑO E IMPLEMENTACIÓN

2.1. Introducción

El presente capítulo abarca cinco fases de la aplicación de la metodología XP al proceso de desarrollo de la propuesta de solución (exploración, planificación de la entrega, iteraciones, producción y mantenimiento). Como parte de tales fases, se realiza una identificación de los requisitos, se obtiene, describe y modela la propuesta de solución y finalmente se planifica y ejecuta la implementación de la misma.

2.2. Fase de exploración

Esta fase permite enmarcar el alcance del proyecto que tiene como fin el desarrollo y la entrega del sistema requerido. Para ello los clientes definen sus necesidades (requisitos) a través de las historias de usuario. Además, durante esta fase se define una propuesta de arquitectura del sistema (31).

2.2.1. Identificación de requerimientos

2.2.1.1. Requerimientos funcionales

Los requerimientos funcionales son capacidades o condiciones que un sistema determinado debe cumplir. Seguidamente se enumeran los que se han identificado para el desarrollo de la propuesta de solución.

RF 1: Crear conexión a base de datos

RF 2: Mostrar tablas de base de datos

RF 3: Mostrar columnas

RF 2: Adicionar columna al esquema de transformación

RF 3: Adicionar columna a un par

RF 4: Integrar datos

RF 5: Mostrar datos integrados

RF 6: Transformar datos integrados a formato CSV

RF 7: Cargar datos

Nota: La notación RF corresponde a Requerimiento Funcional.

2.2.1.2. Requerimientos no funcionales

Los requerimientos no funcionales detallan las propiedades o cualidades que el producto debe tener, aumentándole funcionalidad al sistema, pues hacen al producto atractivo, fácil de usar, rápido y confiable. Los requisitos no funcionales se encuentran separados por categorías.

RNF 1: Apariencia o interfaz externa

- Se deben utilizar imágenes y colores identificados con la herramienta OpenRefine
- La interfaz gráfica de usuario debe estar diseñada para visualizarse en cualquier resolución

RNF 2: Legalidad

- Las herramientas seleccionadas deberán estar respaldadas por licencias permisivas, bajo las condiciones de software libre

RN3: Software

Para el cliente:

- Navegador web

Para el servidor:

- Sistema Operativo Windows, Linux o Mac en cualquiera de sus versiones (o distribuciones, en caso de Linux)
- Postgres SQL en cualquiera de sus versiones
- Máquina Virtual de Java (JVM)

RNF 4: Hardware

Para el cliente:

- Se recomienda como mínimo: procesador Core 2 Duo, 2.2 GHz, 2 GB de memoria RAM

Para el servidor:

- Se recomienda como mínimo: procesador Core 2 Duo, 2.2 GHz, 2 GB de memoria RAM

Nota: La notación RNF corresponde a Requerimiento No Funcional.

2.2.2. Obtención de la propuesta de solución

2.2.2.1. Método teórico

El punto clave en la propuesta de solución es la inclusión de un Procesador de Consultas Distribuidas (DQP). Para lograr una mayor compatibilidad con la solución, se desarrolló un DQP, a partir de la filosofía de trabajo del OGSA-DQP.

A continuación se muestra un flujo de trabajo de alto nivel típico de la solución:

1. Se realiza la alineación de las bases de datos relacionales.
2. Se conforma el esquema de transformación y los pares de columnas a partir de las especificaciones del usuario.
3. Se realiza una consulta global a partir de las columnas incluidas por el usuario en el esquema de trasformación.
4. El DPQ recibe y particiona la consulta global en consultas locales, las cuales distribuye a cada una de las bases de datos locales.
5. Cada una de las bases de datos locales evalúa su correspondiente consulta y retorna un resultado local.
6. El DPQ integra los resultados locales en un resultado global.
7. El resultado global se transforma a un formato admisible por OpenRefine.
8. OpenRefine toma el control realizando el flujo de trabajo básico para la limpieza, transformación y exportación de datos en grafos RDF.

Para mejor comprensión del método teórico obtenido como parte de la propuesta de solución ver Figura 33.

Definición 2: Un esquema de transformación es básicamente una estructura intermedia entre las bases de datos locales y la vista integrada final. El esquema de transformación contiene las columnas de cada una de las bases de datos locales seleccionas por el usuario para conformar la vista final integrada.

Definición 3: Un par de columnas especifica una relación de igualdad entre dos columnas de bases de datos y/o tablas diferentes.

A partir de su funcionamiento puede identificarse en la solución un enfoque GAV en la alineación de las bases de datos relacionales hacia el esquema global, puesto que la forma de proceder se define básicamente a partir de una función que devuelve el resultado de una consulta global sobre el esquema de recursos.

A partir de la implementación particular de la solución, en la que se conforma un esquema de transformación que contiene las columnas de la vista resultante, puede identificarse un enfoque ETL, puesto que las trasformaciones de adición y emparejamiento de columnas se realizan en tiempo real e influyen directamente en la vista resultante.

2.2.2.2. Diseño del Procesador de Consultas Distribuidas

Algunos de los módulos DQP implementados en la actualidad son los mencionados en la sección 1.6.3. Para el propósito de la presente solución, se hizo necesario el uso de un módulo DQP que pudiera ser fácilmente integrable a la misma. Para lograr un mayor nivel de acoplamiento, se decidió realizar la implementación de un nuevo módulo DQP. Para alcanzar tal objetivo se siguió la arquitectura general de OGSA-DQP, con énfasis en la optimización de los procesos de integración a partir de la identificación de los escenarios para los cuales debe tener respuesta la solución que se presenta.

La vista de alto nivel de la arquitectura de un DQP se muestra en la Figura 1.

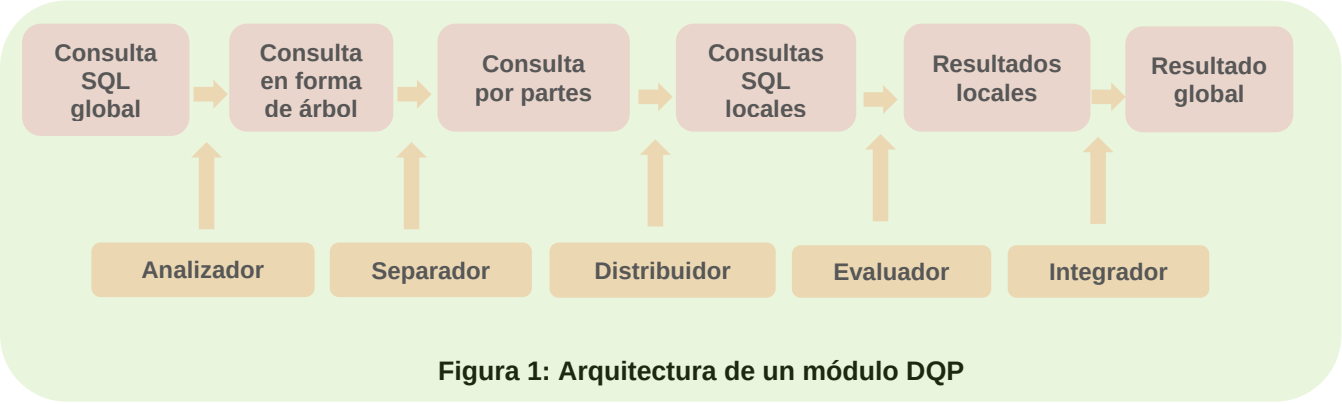

Figura 1: Arquitectura de un módulo DQP

En el caso del DQP que se implementó para la solución se fusionaron el Analizador Léxico y el Separador y los componentes quedaron definidos de la siguiente manera:

1. Parser
2. Distribuitor
3. Evaluator
4. Integrator

Se usó el idioma inglés para lograr uniformidad con el sistema al que se le implementa la extensión. El algoritmo de un módulo DQP para el procesamiento de una consulta global se explica a continuación.

El módulo DQP toma una consulta global como su entrada. La consulta global es procesada por el componente Analizador léxico y transformada en un árbol que representa las tres principales partes de una consulta SQL (SELECT, FROM Y WHERE). El componente Distribuidor, como su nombre lo indica, forma las consultas locales. El Evaluador ejecuta las consultas locales en sus respectivas bases de datos locales. Los resultados de la ejecución de las consultas locales (resultados locales) son integrados por el componente Integrador en un resultado global que constituye, a su vez, el resultado del módulo DQP.

A continuación se muestra una formalización teórica para el algoritmo antes descrito, haciendo uso de los elementos del lenguaje SQL y del Álgebra Relacional.

I. Una consulta SQL $Q_G = (S, F, W)$ es una consulta de la forma:

$SELECT\ \emptyset_1, \dots, \emptyset_n$

$FROM\ T_1, \dots, T_m$

$WHERE\ \theta_1, \dots, \theta_l$

A partir del Álgebra Relacional, puede escribirse una consulta SQL como $\pi\emptyset(\sigma\theta(T))$ donde:

$\phi = \emptyset_1, \dots, \emptyset_n$ donde cada $\emptyset_i$ representa una expresión contenida en el SELECT

$T = T_1, \dots, T_m$ donde cada T_i representa una tabla involucrada en la consulta global

$\theta = \theta_1, \dots, \theta_l$ donde cada $\emptyset_i$ representa una expresión contenida en el WHERE

II. Seguidamente se prosigue a transformar la consulta global en un conjunto de consultas locales. Cada consulta local $Q_{D_m T_n} \in Q_L$ corresponde a una tabla $T_n$que pertenece a una base de datos D_m. Los elementos de la consulta local quedan definidos como:

$\emptyset_{D_m T_n} = \{t.f | d.t.f \in \phi \cup d.t.f \in \theta\}$

$T_{D_m T_n} = \{t | d.t \in T_n\}$

$\theta_{D_m T_n} = \{\}$

Nota: $t.f$ se refiere a la nomenclatura para referirse a un campo específico de una tabla $table.field.\ d.t.f$ se refiere a la nomenclatura para referirse a un campo específico de una tabla dentro de una base de datos determinada.

El Álgebra Relacional para cada consulta local se define como:

$\pi\emptyset_{D_m T_n}(T_n)$

III. El siguiente paso consiste en ejecutar cada una de las consultas locales $Q_{D_m T_n}$ en sus respectivas bases de datos D_m. Cada consulta local queda formada como:

$SELECT\ \emptyset_{D_m T_n}\ FROM\ T_n\ WHERE\ \theta_{D_m, T_n}$

IV. Los resultados de la ejecución de cada una de las consultas locales en sus respectivas bases de datos se almacenan como tablas en una base de datos mediadora M'. De esta forma, por cada consulta local $Q_{D_m T_n} \in Q_L$, una nueva tabla $D_m\_T_n$ con sus columnas $\emptyset_{D_m T_n}$ es creada en M'. Como el nombre de cada nueva tabla tiene la forma d_t (referencia a $database.table$), el nombre de cada columna es de la forma $d\_t\ .f$ (referencia a $database.table.field$).

V. Una vez creada la base de datos mediadora M' con sus correspondientes columnas, una nueva consulta es creada.

$\pi\emptyset'(\sigma\theta'(T'))$ donde:

$\phi' = \{\emptyset'_i | \emptyset'_i = finalTransformPhi(\emptyset_i), \emptyset_i \epsilon \phi\}$

$T' = nuevo\ conjunto\ de\ tablas\ credas\ en\ el\ paso\ anterior$

$\theta' = \{\theta'_i | \theta'_i = finalTransformTheta(\theta_i), \theta_i \epsilon \theta\}$

Las funciones $finalTransformPhi(\emptyset_i)$y $finalTransformTheta(\theta_i)$ reemplazan los identificadores de los atributos en el SELECT y el WHERE respectivamente, con otros de la forma $d\_t\ .f$

La correspondiente consulta a ejecutar sobre la base de datos mediadora M' es de la forma:

$SELECT\ \emptyset'_1, \dots, \emptyset'_n$

$FROM\ T'_1, \dots, T'_m$

$WHERE\ \theta'_1, \dots, \theta'_l$

2.2.3. Diseño de las Historias de Usuario

A partir de la identificación de los requerimientos funcionales se realizó la identificación y diseño de las historias de usuario. A continuación se relacionan las historias de usuario identificadas.

HU 1: Gestionar conexión a bases de datos

HU 2: Gestionar bases de datos

HU 3: Gestionar esquema de transformación

HU 4: Gestionar pares de columnas

HU 5: Integrar datos

HU 6: Cargar datos integrados

Nota: La notación HU se refiere a Historia de Usuario.

A continuación se muestra el diseño de la HU 1 (ver Tabla II). El resto de las historias de usuario se encuentra en el expediente del proyecto y en los anexos del presente documento.

Tabla II: Historia de Usuario 1

HISTORIA DE USUARIO			
Código:	1	**Nombre:**	Gestionar conexión a Bases de Datos
Usuario:	Cliente	**Tipo de actividad:**	Nueva
Riesgo:	Alto	**Prioridad:**	Alta
Iteración:	1	**Puntos estimados:**	2
Descripción:		Cuando el usuario accede al sistema se activa un formulario que contiene en sus campos de entrada los datos necesarios para crear una conexión a base de datos. Al completar la información se crea una conexión a la base de datos especificada. Cada operación que se realice sobre los datos alojados en las bases	

	de datos, es preparada y ejecutada a partir de la clase de acceso a datos.

2.2.4. Diseño de la propuesta de solución

2.2.4.1. Arquitectura

OpenRefine es una aplicación web. Su arquitectura propone una marcada separación entre lo concerniente a los datos y lo relativo a la interfaz gráfica de usuario. De esta forma, el lado del servidor maneja las operaciones de modelado, almacenamiento y transformación de datos, mientras el lado del cliente se ocupa de la construcción de las interfaces gráficas de usuario.

Toda la implementación del lado del servidor se realiza utilizando Java como lenguaje de programación y se ejecuta sobre el servidor web Jetty. El lado del cliente está implementado en Javascript y utiliza jQuery y jQueryUI.

Atendiendo a las características arquitectónicas de OpenRefine, la implementación de la propuesta de solución también siguió una arquitectura Cliente-Servidor.

La Figura 2 muestra una vista de la arquitectura de la propuesta de solución.

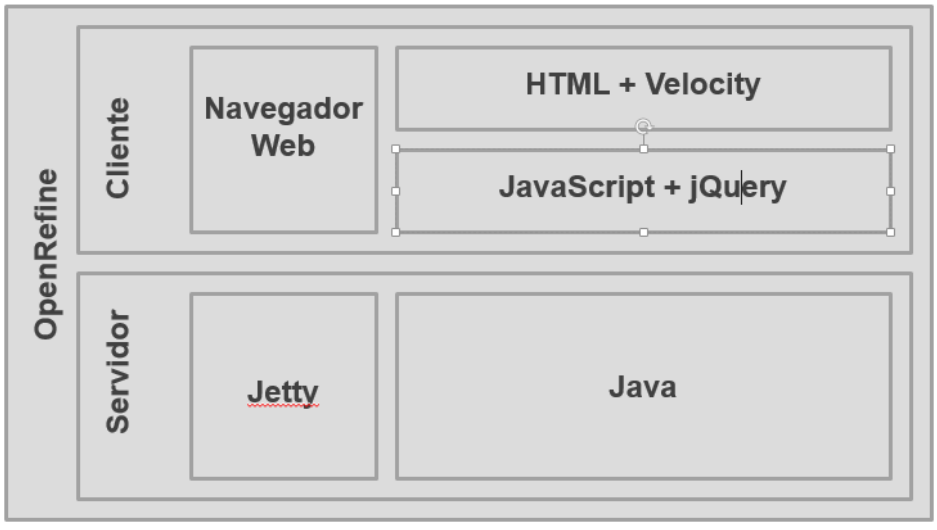

Figura 2: Vista de la arquitectura de la propuesta de solución

2.2.4.2. Estructura del marco de trabajo

La extensibilidad de OpenRefine está soportada a partir del marco de trabajo Butterfly. De esta manera, las extensiones en OpenRefine son módulos de Butterfly. Estos módulos utilizan Java como lenguaje de programación del lado del servidor, en tanto funcionan como una aplicación web del lado del cliente. Una extensión de OpenRefine contiene los siguientes archivos y subdirectorios:

```
build.xml
  src/
      com/foo/bar/... *.java
  module/
     *.html, *.vt
      scripts/... *.js
      styles/... *.css y *.less
      images/... archivos de imágenes
      MOD-INF/
          lib/*.jar
          classes/... clases de java compiladas
          module.properties
          controller.js
```

El código Java (del lado del servidor) está contenido en el subdirectorio **“src/”**. El subdirectorio **“MOD-INF/”** contiene los metadatos de la extensión como módulo del framework Butterfly. El código Java es además compilado dentro del subdirectorio **“classes/”** dentro de **“MOD-INF/”**. El soporte externo para Java contenido en archivos con extensión .jar se ubica en el subdirectorio **“lib/”**, también dentro de **“MOD-INF/”**.

El código del lado del cliente está contenido dentro del subdirectorio **“module/”**. Este código puede estar formado por archivos .html, .css, .js, imágenes o también por archivos LESS, que son procesados dentro del CSS y archivos Velocity (.vt), los cuales necesitan ser especificados en **“MOD-INF/controller.js”**.

El archivo **“MOD-INF/controller.js”** permite la configuración de la inicialización de la extensión.

2.2.4.3. Patrones de diseño

Un patrón es una solución recurrente a un problema dentro de un contexto dado. Los patrones surgen de la experiencia de seres humanos al tratar de lograr ciertos objetivos, además capturan la experiencia existente y probada para promover buenas prácticas.

Para el caso específico de la propuesta de solución se emplea un grupo de patrones relacionados con el diseño de software, llamados patrones GRASP[19] (siglas en inglés de Patrones Generales de Software para Asignación de Responsabilidades), los cuales describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones.

A continuación se relaciona el uso de los patrones de diseño utilizados a través de ejemplos:

Patrón GRASP Experto:

Asigna una responsabilidad al experto en información, la clase que cuenta con la información necesaria para cumplir la responsabilidad. Este patrón es utilizado en las clases entidades ya que estas contienen toda la información correspondiente a las bases de datos, tablas y columnas. (Ver Figura 3)

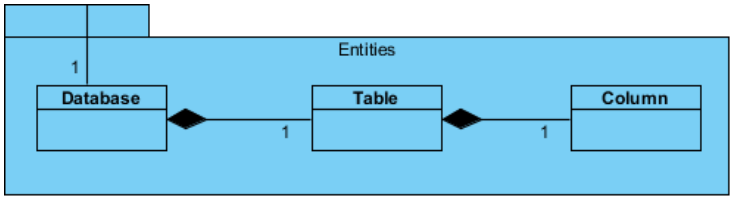

Figura 3: Ejemplo del patrón GRASP Experto

Patrón GRASP Alta Cohesión:

Con este patrón se espera que una clase tenga un número moderado de responsabilidades dentro de un área funcional y colabore con las otras para llevar

[19] Del inglés General Responsibility Assignment Software Patterns

a cabo una tarea. (32) En el caso de la propuesta de solución utilizando el patrón experto se le asigna a cada clase las responsabilidades que le corresponden y se establecen las condiciones para que una clase colabore con las demás en la resolución de tareas que las implican a todas y que no son capaces de resolver por sí solas.

Patrón GRASP Controlador:

Se asigna la responsabilidad del manejo de mensajes de los eventos de un sistema a una clase que representa un manejador artificial de todos los eventos del sistema de un caso de uso que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa.

En la propuesta de solución este patrón se evidencia principalmente en la clase controladora. Todas las peticiones realizadas por el usuario son manejadas por la clase MRDBImportingController.java.

Patrón GRASP Bajo Acoplamiento:

Al programar o diseñar se debe lograr un acoplamiento lo más bajo posible entre dos unidades de software cuales quieran que estas sean. Esto redunda en una mejora considerable en la detección y corrección de errores, en una mayor facilidad de mantenimiento y sobre todo, en la reutilización (33).

En el caso de la propuesta de solución se logra un bajo nivel de acoplamiento entre la extensión implementada y la herramienta OpenRefine. De esta forma, si la extensión es retirada (incluso en tiempo de ejecución), no se altera el funcionamiento de OpenRefine.

Por otra parte, el módulo DQP implementado puede ser utilizado por cualquier otra herramienta escrita en Java que requiera un proceso de integración de datos. La clase Coordinator es el único enlace entre el módulo DQP y la herramienta que haga uso del mismo.

Patrón GRASP Creador:

Este patrón guía la asignación de responsabilidades relacionadas con la creación de instancias, tarea muy frecuente en los sistemas orientados a objetos (33). Para la propuesta de solución, este patrón se utiliza principalmente en las clases controladoras, pues las mismas son las encargadas de la creación de objetos de las entidades. (Ver Figura 4)

```
Writer w = response.getWriter();
JSONWriter writer = new JSONWriter(w);
try {
    writer.object();
    if (exceptions.size() == 0) {
        job.project.update(); // update
```

Figura 4: Ejemplo del uso del patrón GRASP Creador

2.2.4.4. Diagrama de clases

El diagrama de clases del diseño describe gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación y contiene información acerca de las clases, asociaciones, atributos, métodos y dependencias implicados. Estos diagramas son utilizados durante el proceso de análisis y diseño de los sistemas con el objetivo de modelar los aspectos estáticos y definir una solución (34).

En el diseño de la propuesta solución se realizó el diagrama de clases del diseño a partir del uso de estereotipos web. Para su confección fue necesario identificar todas las clases que participan en la propuesta de solución. Para un mejor entendimiento, la Figura 5 muestra el diagrama de clases del diseño obtenido.

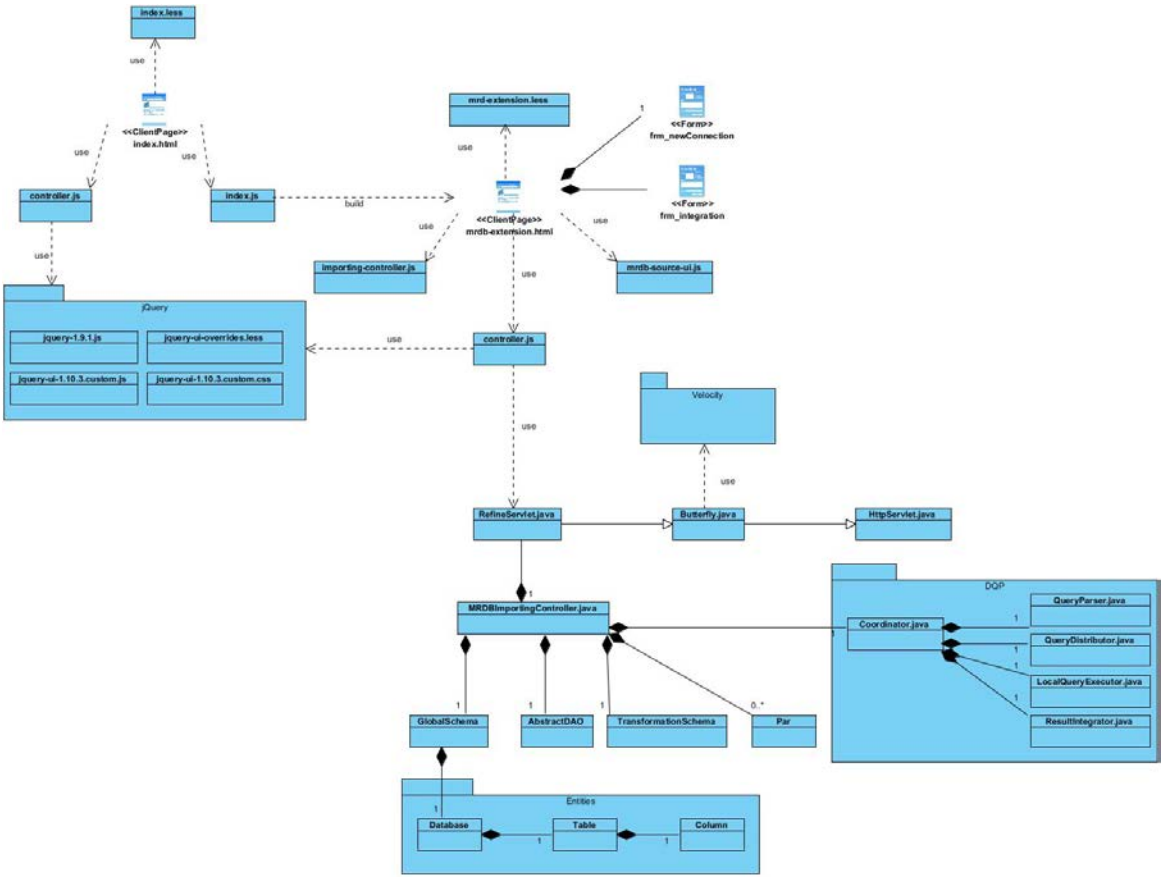

Figura 5: Diagrama de clases del diseño

2.2.4.5. Diagrama de Procesos de Negocio

Un diagrama de procesos de negocio (DPN) es una notación gráfica estandarizada que permite el modelado de procesos de negocio, en un formato de flujo de trabajo (35).

Para lograr un mejor entendimiento de los procesos involucrados en el método teórico obtenido como parte de la propuesta de solución, se diseñó el diagrama de procesos de negocio que se muestra en la Figura 6.

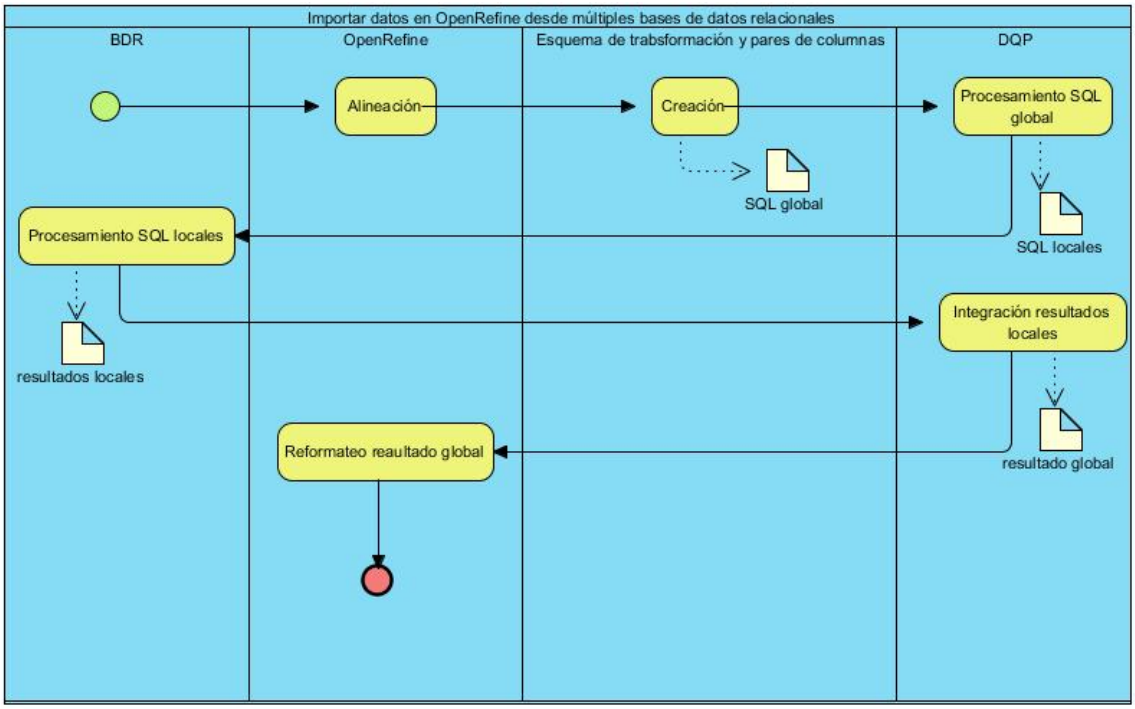

Figura 6: Diagrama de procesos de negocio

2.2.4.6. Diagrama de despliegue

Los diagramas de despliegue se utilizan para modelar la vista de despliegue estática. Además describen la configuración del sistema para su ejecución en un ambiente del mundo real (34). La Figura 7 muestra el diagrama de despliegue obtenido como parte de la modelación de la propuesta de solución:

Figura 7: Diagrama de despliegue de la propuesta de solución

La vista de despliegue está conformada por un nodo PC Cliente que representa las estaciones de trabajo desde donde se conectan los clientes o usuarios al servidor

Jetty usando el protocolo HTTP. En dicho servidor se encuentra desplegada la aplicación.

2.2.5. Estándares de codificación

Un estándar de codificación comprende todos los aspectos relacionados con la generación de código, de tal manera entendible para todos los programadores. Un código fuente completo debe reflejar un estilo armonioso y uniforme, como si un único programador hubiera escrito todo el código. El método para lograr que un grupo de desarrolladores mantenga un código de calidad es establecer un estándar de codificación.

Los estándares utilizados en la codificación del componente fueron los siguientes:

Identificadores:

Se utilizó el estilo UperCamelCase para:

- Nombrar las clases. (Ver Figura 8)

```
public class QueryParser {

    public QueryParser() {...2 lines }

    public SFWQueryAST buildQueryAST(String query) {...12 lines }
```

Figura 8: Estilo UperCamelCase para las clases

- Nombrar las funciones. (Ver Figura 9)

```
public WhereAST createWhereAST(String whereClause) {...61 lines }
```

Figura 9: Estilo UperCamelCase para las funciones

Se utilizó el estilo lowerCamelCase para:

- Nombrar las variables. (Ver Figura 10)

```
int indexListSelect = query.lastIndexOf("SELECT") + 7;
int indexFromClause = query.lastIndexOf("FROM") + 5;
int indexWhereClause = query.lastIndexOf("WHERE") + 6;
```

Figura 10: Estilo lowerCamelCase para las variables

Identación:

Se definió como estilo de Identación el K&R con tabulación de 7 espacios.

Comentarios:

Se definió el uso de comentarios claros y precisos que ayudarán a una mejor comprensión del código implementado.

- Encima de las funciones. (Ver Figura 11)

```
 //Creacion de la base de datos mediadora
public void createMediatorDataBase(LinkedList<Response> localResults, Ab

//Construir consulta a la base de datos mediadora
public String finalTransformSelect(SelectAST listSelect) {...9 lines }
```

Figura 11: Comentarios encima de las funciones

- Dentro del cuerpo de las funciones. (Ver Figura 12)

```
String where = " WHERE ";
for (int i = 0; i < whereClause.getWhereItems().size(); i++) {
    //Para el operador1
    where = where + comilla.toString() + whereClause.getWhereItems
    //Para el operador2
    where = where + comilla.toString() + whereClause.getWhereItems
}
//Quitar el ultimo AND
String finalWhere = where.substring(0, where.length() - 4);
return finalWhere;
```

Figura 12: Comentarios en el cuerpo de las funciones

Líneas y espacios en blanco:

- Entre funciones. (Ver Figura 13)

```
public SFWQueryAST buildQueryAST(String query) {...12 lines }

public SelectAST createSelectAST(String listSelect) {...40 lines }

public FromAST createFromAST(String fromClause) {...29 lines }

public WhereAST createWhereAST(String whereClause) {...61 lines }
```

Figura 13: Líneas en blanco entre funciones

- Entre implementaciones dentro del cuerpo de las funciones. (Ver Figura 14)

```
LinkedList<Attribute> attributeList = new LinkedList<Attribute>();
SelectAST items = new SelectAST(attributeList);

String database = "";
String table = "";
String column = "";
```

Figura 14: Líneas en blanco dentro del cuerpo de las funciones

Llaves:

En la codificación de la propuesta de solución las llaves de apertura de las funciones se colocaron en la misma línea de declaración de la orden, lo mismo que las llaves de aperturas de las estructuras for, if, while, else. Las llaves de cierre se colocaron solitarias en la línea que sigue a la última línea dentro del bloque e identadas al nivel de la línea cabecera del bloque. La ventaja de usar este estilo es que las llaves iniciales no necesitan ninguna línea extra para ellas solas, por lo que se ahorra espacio vertical de lectura.

2.3. Fase de planificación de la entrega

En esta fase el cliente establece la prioridad de cada historia de usuario. Los programadores realizan una estimación del esfuerzo necesario para desarrollar cada una de ellas. Además, como parte de esta fase de genera un plan de entregas atendiendo a la cantidad de iteraciones a realizar.

2.3.1. Estimación del esfuerzo por Historia de Usuario

De acuerdo a la prioridad asignada por el cliente a cada historia de usuario y teniendo en cuenta la complejidad y riesgo determinados por el programador, se realiza la estimación de cada una de las historia de usuario identificadas. La unidad de estimación es el punto. Un punto equivale a una semana ideal de programación.

La siguiente tabla muestra la estimación de esfuerzo por historia de usuario realizada.

Tabla III: Estimación de esfuerzo por HU

Historia de Usuario	Puntos de estimación
Gestionar conexión a Bases de Datos (HU1)	2
Gestionar Bases de Datos (HU2)	2
Gestionar esquema de transformación (HU3)	3
Gestionar pares de columnas (HU4)	3
Integrar datos (HU5)	4
Cargar datos integrados (HU6)	3

Para realizar una planificación de las entregas, previamente se genera un plan de iteraciones. A continuación se muestran el plan de iteraciones y el plan de entregas obtenidos. (Ver Tabla IV y Tabla V)

Tabla IV: Plan de iteraciones

Iteración	Número de la historia	Historia de usuario	Duración (semanas)	
Iteración # 1	01	Gestionar conexión a Bases de Datos	2	10
	02	Gestionar Base de datos	2	
	03	Gestionar esquema de trasformación	3	
	04	Gestionar pares de columnas	3	
Iteración # 2	05	Gestionar integración	4	7
	06	Cargar datos integrados	3	

Tabla V: Plan de entregas

Iteración	Iteración #1	Iteración #2
Cantidad de historias de usuario	4	3
Fecha de entrega	27/03/2014	15/05/2014

2.4. Fase de iteraciones

Como parte de la fase de iteraciones se dará cumplimiento al plan de iteraciones previamente diseñado en la fase de planificación. (Ver Tabla IV)

Para dar cumplimiento al plan de iteraciones, se ha de tener en cuenta el conjunto de tareas de ingeniería previamente identificado y diseñado durante la fase de planificación. (Ver anexo 2)

2.5. Fase de producción

En esta fase se define el modelo de dominio a través de las clases CRC priorizando la orientación a objetos sobre el enfoque procedimental.

2.6. Tarjetas Clase-Responsabilidad-Colaboración (CRC)

Una de las tareas más importantes en el diseño de software lo constituye el definir correctamente el modelo de dominio (conjunto de clases de la lógica de negocio y sus interrelaciones) con que se trabaja. En XP, a diferencia de otras metodologías de desarrollo de software, no es necesario definir un modelo de dominio (diagramas de clases UML[20]). En su lugar se propone el uso de tarjetas Clase-Responsabilidad-Colaboración (CRC), técnica que ayuda a evitar el enfoque procedimental destacando la orientación a objetos (36).

A continuación se muestra la tarjeta CRC de la clase MRDBImportingController.java, una de las más importantes en la implementación de la propuesta de solución. (Ver

[20] Siglas en inglés de Unified Modeling Language

Tabla VI). El resto de las tarjetas CRC se encuentra en el expediente del proyecto y en los anexos del presente documento. (Ver anexo 3)

Tabla VI: Tarjeta CRC clase MRDBImportingController.java

MRDBImportingController.java	
Responsabilidad	**Colaboración**
Procesar los campos de entrada para crear la conexión a base de datos	importing-controller.js mrdb-source-ui.js
Adicionar columnas al esquema de transformación	
Crear pares	

2.7. Control de la seguridad

Como parte de la fase de producción, se define una técnica para el aseguramiento de la seguridad. Como ni OpenRefine ni la extensión implementada hacen uso de roles, usuarios o perfiles, el aseguramiento de la seguridad se centra en evitar la infiltración de código, especialmente a través de inyecciones SQL.

Una inyección SQL es un método de infiltración de código intruso que se vale de una vulnerabilidad informática presente en una aplicación en el nivel de validación de las entradas para realizar consultas a una base de datos.

Con el objetivo de evitar este tipo de violaciones de la seguridad, durante la implementación de la propuesta de solución se aplicaron dos técnicas, las cuales se describen a continuación:

- Uso de la clase PreparedStatement en lugar de:

Statement stmt = conn.createStatement();

ResultSet rset = stmt.executeQuery(“SELECT * FROM book WHERE title =” + title);

- Parametrización de sentencias SQL, como se indica a continuación

```
PreparedStatement pstmt = conn.prepareStatement("SELECT * FROM book WHERE title = ?");

pstmt.setString(1, title);

ResultSet rset = pstmt.executeQuery();
```

2.8. Fase de mantenimiento

Con el objetivo de dedicar el capítulo 3 del presente documento a la realización de las pruebas de software en correspondencia con la fase de pruebas propuesta por la metodología XP, a continuación se describen las fases de mantenimiento y muerte del proyecto.

El mantenimiento de software es una de las actividades más comunes en la ingeniería de software. Consiste en un proceso de mejora y optimización del software después de su entrega al usuario final (es decir; revisión del programa), así como también corrección y prevención de los defectos.

Existen cuatro tipos de mantenimiento, los cuales son: (34)

- Perfectivo
- Evolutivo
- Adaptativo
- Correctivo

Dentro de las principales técnicas de mantenimiento se encuentran (37):

- Reingeniería
- Ingeniería inversa
- Reconstrucción del software

Con el objetivo de llevar a cabo la fase de mantenimiento se creará un equipo que tendrá la responsabilidad de realizar cada uno de los tipos de mantenimiento mencionados.

La Figura 15 describe el proceso que se realizará como parte de la ejecución del mantenimiento y está basada en el método propuesto en (37).

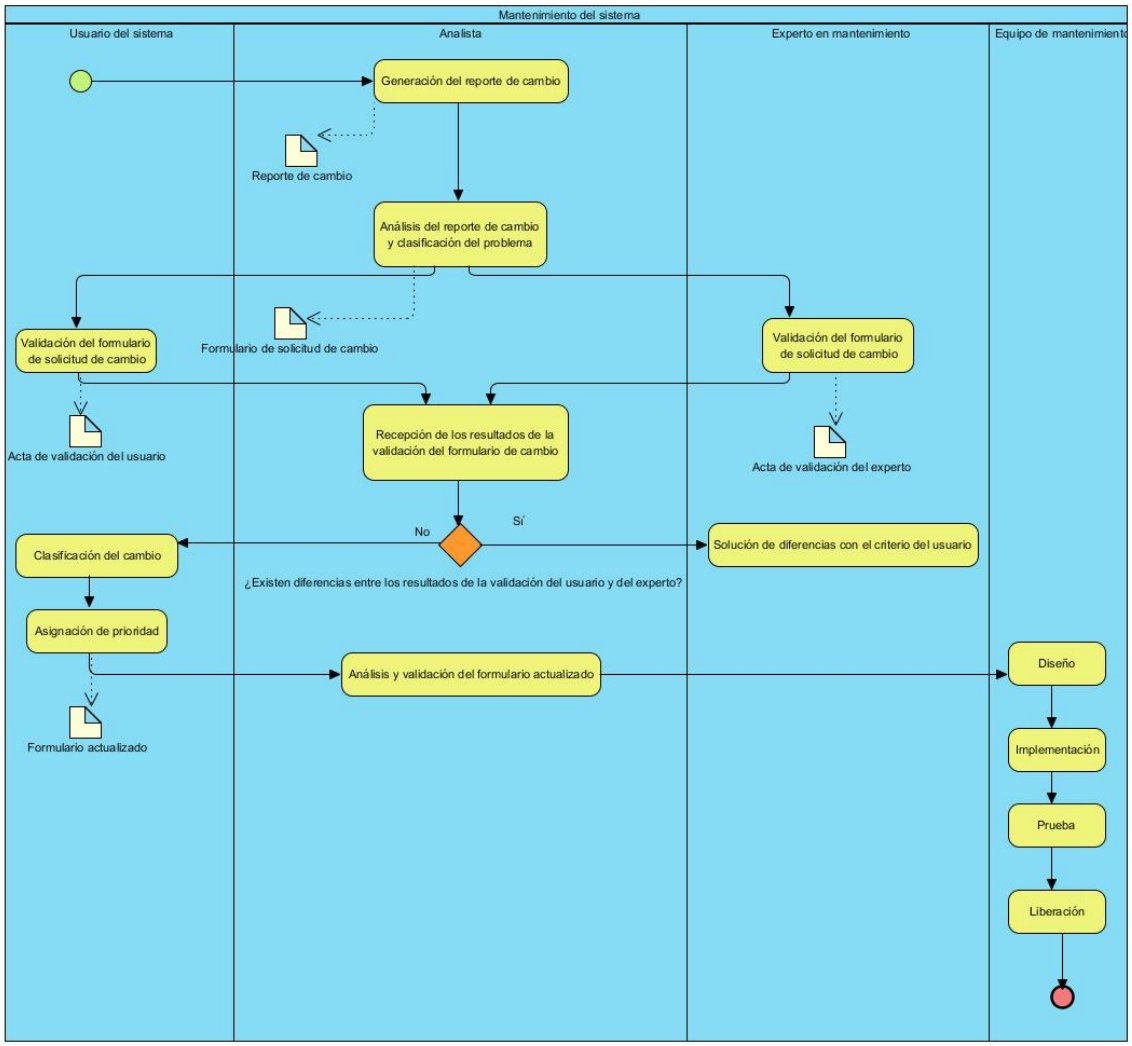

Figura 15: Diagrama de procesos del negocio referente al mantenimiento

Inicialmente el usuario de la aplicación experimenta un deseo de cambio sobre uno o varios elementos de la misma y genera un reporte donde ha de mostrar la naturaleza del deseo de cambio (problema funcional, no funcional u otro tipo). Seguidamente, un miembro del grupo de mantenimiento (analista) analiza el reporte e identifica si se trata de un error de software y si es factible la realización del cambio en el sistema. Un cambio se considerará factible cuando suponga un mejoramiento de las funcionalidades del sistema y/o cuando se precise del mismo por variaciones del entorno (de software y/o de hardware). Además, este miembro del grupo de

mantenimiento deberá realizar una clasificación del problema especificado en el reporte. Las clasificaciones posibles son software y hardware.

Luego de realizar tal clasificación se genera un formulario que incluye: estimación del esfuerzo requerido para la realización del cambio, recursos computacionales requeridos e impacto del cambio en la calidad del sistema. El analista debe llenar cada uno de los campos del formulario y enviarlo al usuario de la aplicación para recibir su validación. Además, este formulario es enviado al miembro del grupo experto en mantenimiento de software, quien también realiza una evaluación del mismo y ante diferencias con la valoración emitida por el usuario, este es el encargado de darles solución.

Seguidamente, el usuario clasifica el cambio a partir del formulario en modificación o reparación. De acuerdo a la clasificación otorgada por el usuario posteriormente el equipo de mantenimiento decide la técnica a usar. Una modificación se refiere a la inclusión de una nueva funcionalidad que no se encontraba especificada en los requerimientos iniciales, mientras una reparación se refiere a la corrección de una o varias funcionalidades del sistema. Además, el usuario especifica un grado de prioridad que puede ser (Emergencia si el cambio es de alta prioridad, Urgente si el problema impide el desempeño de las funcionalidades principales del sistema o Rutinario para otros casos).

Una vez clasificado el cambio por el usuario, el formulario correspondiente es analizado y validado por el analista y finalmente pasa al equipo de mantenimiento donde se realiza el diseño, la codificación y la prueba del cambio realizado con la consecuente liberación de la nueva versión del sistema.

2.9. Fase de muerte del proyecto

Una vez que el cliente no tenga más historias de usuario para ser incluidas en el sistema y se hayan satisfecho sus necesidades en otros aspectos como rendimiento y confiabilidad del sistema, se genera la documentación final del sistema y no se realizan más cambios en la arquitectura (31). Este punto se conoce como muerte del proyecto.

Para la propuesta de solución, aun no se ha llegado a la muerte del proyecto, puesto que con el desarrollo de la implementación del método teórico propuesto se ha obtenido un prototipo funcional, el cual debe ser mejorado a partir de la aplicación del mantenimiento.

2.10. Conclusiones del capítulo

Las historias de usuario descritas por el cliente definen los requisitos que debe cumplir el sistema, así como el nivel de detalle para cada funcionalidad. Por otro lado, la arquitectura del sistema permite la mantenibilidad y escalabilidad.

La planificación de las iteraciones y entregas permiten la organización de las Historias de Usuario de acuerdo a la prioridad del cliente y su posterior separación en tareas de ingeniería de acuerdo al tiempo requerido para su implementación.

Las tarjetas CRC determinan el grado de acoplamiento del sistema, al definir las clases y sus responsabilidades, así como la colaboración entre ellas.

CAPÍTULO 3. EVALUACIÓN DE LA PROPUESTA DE SOLUCIÓN

3.1. Introducción

Uno de los pilares de XP es el proceso de pruebas. Este permite aumentar la calidad del sistema reduciendo el número de errores no detectados y disminuyendo el tiempo transcurrido entre la aparición de un error y su detección. También evitan efectos colaterales no deseados a la hora de realizar modificaciones y refactorizaciones.

XP divide las pruebas del sistema en dos grupos: pruebas unitarias, encargadas de verificar el código y diseñada por los programadores, y pruebas de aceptación o pruebas funcionales destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida diseñada por el cliente final (38).

Pruebas unitarias

Las pruebas unitarias están enfocadas a probar los elementos más pequeños del software. Es aplicable a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos, y que ellos funcionen como se espera. La prueba de unidad siempre está orientada a caja blanca.

Las pruebas de caja blanca se basan en el minucioso examen de los detalles procedimentales. Se comprueban los caminos lógicos del software proponiendo casos de prueba que examinen que están correctas todas las condiciones y/o bucles para determinar si el estado real coincide con el esperado o afirmado. Esto genera gran cantidad de caminos posibles por lo que hay que dedicar esfuerzos a la determinación de las condiciones de prueba que se van a verificar.

Este tipo de prueba será aplicada a la estructura procedimental (código fuente) de las funcionalidades que implementa cada historia de usuario, a través de la técnica del camino básico.

La prueba del camino básico es una técnica de prueba de caja blanca propuesta por Tom McCabe. Esta técnica permite obtener una medida de la complejidad lógica

de un diseño y usar esta medida como guía para la definición de un conjunto básico. (39)

La idea es derivar casos de prueba a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control. Para obtener dicho conjunto de caminos independientes se construye el grafo de flujo asociado y se calcula su complejidad ciclomática. Los pasos que se siguen para aplicar esta técnica son:

1. A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
2. Se calcula la complejidad ciclomática del grafo.
3. Se determina un conjunto básico de caminos independientes.
4. Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

Los casos de prueba derivados del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

Pruebas de aceptación

Las pruebas de aceptación son creadas a partir de las historias de usuario. Durante una iteración la historia de usuario seleccionada en la planificación de iteraciones se convertirá en una prueba de aceptación. El cliente o usuario especifica los aspectos a probar cuando una historia de usuario ha sido correctamente implementada.

Una prueba de aceptación es como una caja negra. Cada una de ellas representa una salida esperada del sistema. Es responsabilidad del cliente verificar la corrección de las pruebas de aceptación y tomar decisiones acerca de las mismas (38).

Para implementar este nivel de prueba se utilizaran pruebas de caja negra, creando para cada historia de usuario uno o más casos de prueba en dependencia de las funcionalidades que involucre.

Cada caso de prueba debe contener un código, la historia de usuario a la que pertenece, el nombre, una breve descripción, la acción a probar, los datos de entrada, los resultados esperados y la evaluación de la prueba.

3.2. Pruebas de caja blanca

Al sistema desarrollado se le aplicó la prueba de caja blanca, haciendo uso de la técnica del camino básico, con el objetivo de evaluar la complejidad lógica de un diseño procedimental y usar esta medida como guía para la definición de un conjunto básico de caminos de ejecución. Esta prueba permite garantizar que en los casos de prueba obtenidos a través del camino básico se ejecute cada sentencia del programa por lo menos una vez.

A continuación se analizan y enumeran las sentencias de código del método ***executeLocalQueries*** contenido en la clase ***LocalQueryExecutor.java***. Este método retorna una lista que contiene los resultados de la ejecución de las consultas locales a sus respectivas bases de datos relacionales. (Ver Figura 15)

```
public LinkedList<Response> executeLocalQueries(LinkedList<LocalQuery> localQueries) throws SQLException {
        LinkedList<Response> responses = new LinkedList<Response>(); //1
        for (int i = 0; i < localQueries.size(); i++) { //2
            Character comilla = '"';
            String selectItems = "";
            String item = ""; //3
            for (int j = 0; j < localQueries.get(i).getLocalSelect().getLocalSelect().size(); j++) { //4
                if (j == localQueries.get(i).getLocalSelect().getLocalSelect().size() - 1) { //5
                    item += localQueries.get(i).getLocalSelect().getLocalSelect().get(j).getTable() +
                    "." + localQueries.get(i).getLocalSelect().getLocalSelect().get(j).getField() +
                    " AS " + comilla.toString() + localQueries.get(i).getLocalSelect().getLocalSelect().get(j).getDatabase() +
                    "_" + localQueries.get(i).getLocalSelect().getLocalSelect().get(j).getTable() +
                    "." + localQueries.get(i).getLocalSelect().getLocalSelect().get(j).getField() + comilla.toString(); //6
                } else {
                    item += localQueries.get(i).getLocalSelect().getLocalSelect().get(j).getTable() +
                    "." + localQueries.get(i).getLocalSelect().getLocalSelect().get(j).getField() +
                    " AS " + comilla.toString() + localQueries.get(i).getLocalSelect().getLocalSelect().get(j).getDatabase() +
                    "_" + localQueries.get(i).getLocalSelect().getLocalSelect().get(j).getTable() + "." +
                    localQueries.get(i).getLocalSelect().getLocalSelect().get(j).getField() + comilla.toString() + ", ";//7
                }
            }
            selectItems += item;
            String fromItem = localQueries.get(i).getLocalFrom().getTable();
            String query = "SELECT " + selectItems + " FROM " + fromItem;
            AbstractDAO cnx = new AbstractDAO(localQueries.get(i).getDatabase(), "localhost", "postgres", "postgres", 5432);
            ResultSet result = cnx.execute(query);
            String tableName = localQueries.get(i).getLocalFrom().getDatabase() + "_" + localQueries.get(i).getLocalFrom().getTable();
            Response r = new Response(result, tableName);
            responses.add(r); //8
        }
        return responses; //9
    }
```

Figura 16: Función que retorna el conjunto de resultados de la ejecución de las consultas locales a sus respectivas bases de datos relacionales

Después de este paso, es necesario representar el grafo de flujo asociado al código antes presentado a través de nodos, aristas y regiones. (Ver Figura 16)

Una vez construido el grafo de flujo asociado al procedimiento anterior se determina la complejidad ciclomática, la cual es una métrica de software útil pues proporciona una medición cuantitativa de la complejidad lógica de un programa. El valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y da un límite superior para el número de pruebas que se deben realizar.

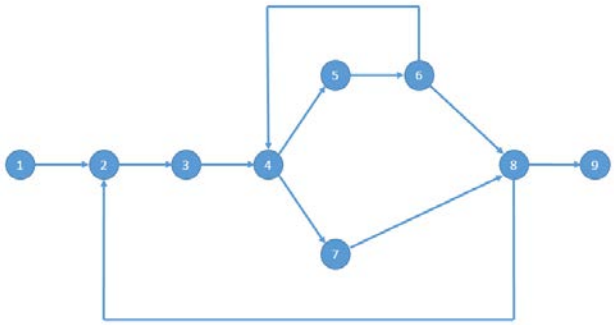

Figura 17: Grafo de flujo asociado al método executeLocalQueries

Cálculo de la complejidad ciclomática para el grafo de flujo de la Figura 7.

$V(g) = (a - n) + 2\ (I)$ $V(g) = p + 1(II)$ $V(g) = r\ (III)$

Sean “V (g)” el valor de la complejidad ciclomática, “a” la cantidad total de aristas, “n” la cantidad total de nodos, “p” la cantidad total de nodos predicados (nodos de los cuales parten dos o más aristas) y “r” la cantidad total de regiones, incluyendo el área exterior del grafo como una región más.

$V(g) = (11 - 9) + 2 = 4\ (I)$ $V(g) = 3 + 1 = 4(II)$ $V(g) = 4\ (III)$

La evaluación de las fórmulas I, II y III arroja un valor de complejidad ciclomática igual a 4, de manera que existen 4 posibles caminos por donde el flujo puede circular. Este valor representa el número mínimo de casos de pruebas para el procedimiento tratado. Seguidamente, es necesario especificar los caminos básicos que puede tomar el algoritmo durante su ejecución.

Los caminos básicos identificados son:

Camino básico # 1: 1, 2, 3, 4, 7, 8, 9

Camino básico # 2: 1, 2, 3, 4, 5, 6, 4, 7, 8, 9

Camino básico # 3: 1, 2, 3, 4, 7, 8, 2, 3, 4, 7, 8, 9

Camino básico # 4: 1, 2, 3, 4, 8, 8, 2, 3, 4, 5, 6, 4, 7, 8, 9

Se procede a ejecutar los casos de pruebas para cada uno de los caminos básicos determinados en el grafo de flujo. Para definir los casos de prueba es necesario tener en cuenta:

Descripción: Se describe el caso de prueba y se especifican los aspectos fundamentales de los datos de entrada.

Condición de ejecución: Se verifica que cada parámetro cumpla las condiciones de ejecución.

Entrada: Se muestran los parámetros de entrada del procedimiento.

Resultados Esperados: Se especifica el resultado que debe arrojar el procedimiento.

Los casos de prueba diseñados para las pruebas de caja blanca se encuentran en el expediente del proyecto y en los anexos del presente documento. (Ver anexo 4)

Resultados:

Se aplicó un total de 60 casos de pruebas de caja blanca (pruebas unitarias). De ellos, 5 tuvieron resultados no satisfactorios, lo cual representa aproximadamente el 8% del total de casos de prueba de caja blanca aplicados, ver fig. 8. Mientras, los 55 casos de prueba restantes tuvieron resultados satisfactorios para un 92% del total. Los errores detectados a través de la aplicación de los casos de pruebas fueron mitigados después de 2 iteraciones de prueba. (Ver Figura 17)

Figura 18: Distribución de los resultados de la ejecución de los casos de prueba de caja blanca

3.3. Pruebas de Caja Negra

Para la realización de las pruebas de caja negra se empleó la técnica Partición de Equivalencia. Esta permite examinar los valores válidos e inválidos de las entradas existentes en el software.

Los casos de prueba diseñados para las pruebas de caja negra se encuentran en el expediente del proyecto y en los anexos del presente documento. (Ver anexo 4)

Resultados:

Se realizaron un total de 30 casos de pruebas de caja negra (pruebas de aceptación), de ellos 12 tuvieron resultados no satisfactorios, lo cual representa el 40% del total de casos de prueba de caja negra realizados, ver Fig. 9. Mientras los 17 casos de prueba restantes tuvieron resultados satisfactorios para un 60% del total. Los errores detectados a través de la aplicación de los casos de pruebas fueron mitigados después de 2 iteraciones de prueba. (Ver Figura 18)

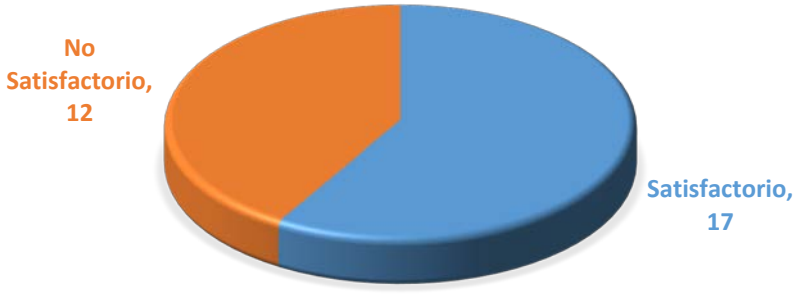

Figura 19: Distribución de la evaluación de los resultados de la aplicación de las pruebas de caja negra a las Historias de Usuario HU-1 y HU-3

3.4. Validación de la propuesta de solución

3.4.1. Entorno de evaluación

Para realizar una correcta validación de la solución se ha de definir primeramente el entorno de evaluación.

La evaluación se realizó en una laptop con las siguientes características:

CPU: Intel(R) Core(TM) 2 Duo, 2.20 GHz

RAM: 2048 MB

Capacidad de almacenamiento: 320 GB

Sistema Operativo: Windows 8 Enterprise

Java: Java SDK. 1.7

Netbeans: NetBeans IDE 7.4

Sistema Gestor de Bases de Datos: PostgresSQL 9.4

Navegador Web: Mozilla Firefox 27.0

3.4.2. Caso de estudio

La evaluación se realiza a través del siguiente caso de estudio, el cual está diseñado de acuerdo al objetivo general de la investigación. El caso de estudio consiste un escenario imaginario relacionado con bibliotecas del mundo, sus catálogos de libros y autores.

A continuación se realiza una **descripción** del caso de estudio:

La Biblioteca Pública de Nueva York, la quinta más grande del mundo, tiene una base de datos formada por los libros pertenecientes a su catálogo. De los libros se conoce el código isbn y el título. (Ver Figura 19)

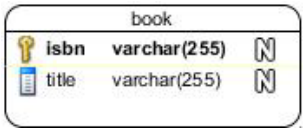

Figura 20: Modelo Entidad Relación de la base de datos “newyork”

Por otra parte, la Biblioteca Británica es la Biblioteca Nacional del Reino Unido, la segunda más grande del mundo. Esta biblioteca tiene una base de datos formada por los libros y casas editoriales pertenecientes a su catálogo, la cual se encuentra disponible en la página oficial de la misma. De los libros se conoce el código isbn, el título, el género y el número de páginas. De las casas editoriales se conoce su nombre y país. (Ver Figura 20)

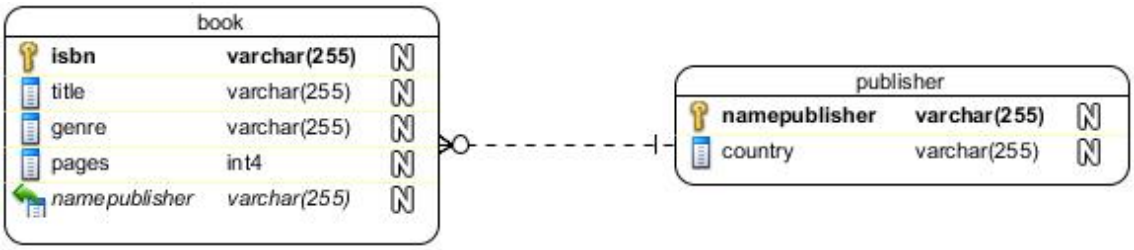

Figura 21: Modelo Entidad Relación de la base de datos “british”

Actualmente todo el soporte web de las bibliotecas de Estados Unidos se encuentra en proceso de migración de la web 2.0 hacia la web 3.0 o Web Semántica, razón por la cual se necesita que toda la información que se encuentra almacenada en bases de datos relacionales sea convertida a grafos RDF. Además, la biblioteca

pretende aumentar la información que posee sobre los libros de su catálogo, para lo cual realiza una descarga de la base de datos de la Biblioteca Británica.

La tabla de libros de la base de datos de la Biblioteca Pública de Nueva York finalmente debería incluir información sobre el código isbn, el título, el número de páginas, y el nombre de la casa editorial.

3.4.3. Aplicación del método obtenido al caso de prueba

1. Se realiza la alineación de las bases de datos relacionales alojadas en el servidor local.

   Primeramente se escriben los datos para la creación de una conexión a la base de datos. Inicialmente se establecerá la conexión a la base de datos “british”, la cual al ser descargada de la página oficial de la Biblioteca Británica, se encuentra disponible en el servidor local postgres. El sistema muestra las tablas pertenecientes a la base de datos a la que se estableció la conexión. Al seleccionar una tabla se muestran sus columnas. (Ver anexo 5, Figura 26 y Figura 27)
2. Se conforman el esquema de transformación y los pares columnas a partir de las especificaciones del usuario.

   Se seleccionan las columnas de las tablas de la base de datos que se desea que formen parte de la vista integrada final y, por cada columna que se marque, se presiona el botón “Add To Select”.

   De la base de datos “british” se desea información sobre el número de páginas y las casas editoriales, por lo que las columnas seleccionadas son “pages” y “namepublisher”, pertenecientes a la tabla “book”.

   Además, se deben seleccionar aquellas columnas por las que se establecerá la relación con la otra base de datos que se desea integrar. Para este caso de estudio, la columna que forma parte de los pares (columnas que representan un mismo concepto en dos bases de datos distintas) es “isbn”.

   Como las restantes columnas que se desean incluir en la vista integrada final se encuentran en la base de datos “newyork”, es necesario establecer una nueva conexión, por lo que se selecciona la opción “New Connection”.

Seguidamente se introducen los datos para crear la conexión a la base de datos “newyork” y se seleccionan las columnas que se desea formen parte de la vista integrada final, siguiendo el mismo procedimiento descrito para la base de datos “british”.

De la base de datos “newyork” se desea información sobre el código isbn y el título, por lo que las columnas seleccionadas son “isbn” y “title”, pertenecientes a la tabla “book”. Seguidamente se debe completar el par que se comenzó a formar cuando se seleccionó en la base de datos “british” la columna “isbn”. Para esto se selecciona en la base de datos actual (“newyork”) la columna “isbn” y se presiona el botón “Add To Par”. De esta manera se completa el par que une a ambas bases de datos a partir de la columna “isbn” contenida en sus respectivas tablas “book”.

3. Seguidamente se genera una consulta global a partir de las columnas incluidas por el usuario en el esquema de trasformación y atendiendo al criterio de igualdad que representan los pares de columnas. (Ver anexo 5, Figura 28)
4. El DPQ recibe la consulta. La clase Coordinator del DQP es la encargada de comunicar OpenRefine (o cualquier otra aplicación) con el mismo y de realizar todos los pasos para la integración. Su método principal es “integrate”, el cual recibe como parámetros una consulta global en forma de cadena de texto y una conexión a base de datos.
5. El componente Parser del DQP transforma la consulta en un árbol como el que muestra la Figura 21.

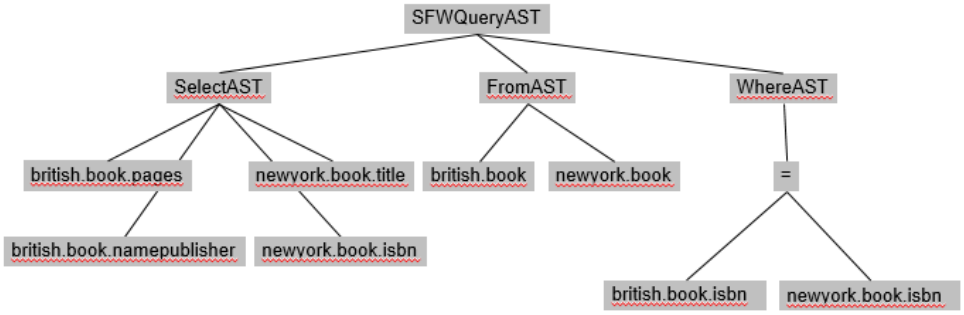

Figura 22: Consulta global en forma de árbol generada por el componente Parser

6. El componente Distributor particiona y distribuye la consulta global como consultas SQL locales

   Para el caso de estudio se generan dos consultas locales, las cuales son:

   (1) SELECT british.book.pages, british.book.namepublisher FROM british.book

   (2) SELECT newyork.book.isbn, newyork.book.title FROM newyork.book

7. El componente Evaluator ejecuta cada una de las consultas locales en sus correspondientes bases de datos y devuelve como resultado un conjunto de resultados locales. (Ver Figura 22 y Figura 23)

pages integer	namepublisher character varying(255)
672	Voyager Books
554	Putnam
324	George Allen & Unwin
136	Hetzel
251	Bloomsbury
450	Doubleday & Company
312	Alfaguara
645	Edhasa
223	Bloomsbury
184	Éditions Gallimard

Figura 23: Resultado local correspondiente a la consulta local (1)

isbn character varying(255)	title character varying(255)
0-399-11697-4	Children of Dune
84-450-7037-1	The Hobbit
84-7888-562-5	Harry Potter and the Chamber of Secrets
970-07-3909-0	Deux ans de vacances
0-00-224584-1	A Game of Thrones
84-01-49932-1	Exodus
84-204-4236-4	Girl With a Pearl Earring
84-350-0447-3	Schindlers Ark
84-7888-561-7	Harry Potter and the Philosophers Stone
987-1144-31-8	L Étranger

Figura 24: Resultado local correspondiente a la consulta local (2)

8. El componente Integrator crea un esquema mediador en el cual se almacenan como tablas cada uno de los resultados locales. (Ver Figura 24)

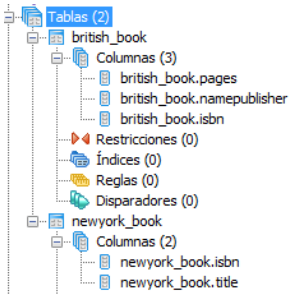

Figura 25: Esquema mediador resultante de las consultas locales (1) y (2)

9. El componente Integrator realiza una consulta local hacia el esquema mediador a partir de un conjunto de transformaciones que se realizan sobre la consulta global inicial.

   La consulta una vez transformada es: SELECT "british_book.pages", "british_book.namepublisher", "newyork_book.isbn", "newyork_book.title" FROM british_book, newyork_book WHERE "british_book.isbn" = "newyork_book.isbn"

   El resultado de la ejecución de la consulta es llamado resultado global. La Figura 25 muestra el resultado global obtenido para el caso de estudio.

All		british_book.pa	british_book.nam	newyork_book.	newyork_book.title
1.	554	Putnam	0-399-11697-4	Children of Dune
2.	324	George Allen & Unwin	84-450-7037-1	The Hobbit
3.	251	Bloomsbury	84-7888-562-5	Harry Potter and the Chamber of Secrets
4.	136	Hetzel	970-07-3909-0	Deux ans de vacances
5.	672	Voyager Books	0-00-224584-1	A Game of Thrones
6.	450	Doubleday & Company	84-01-49932-1	Exodus
7.	312	Alfaguara	84-204-4236-4	Girl With a Pearl Earring
8.	645	Edhasa	84-350-0447-3	Schindlers Ark
9.	223	Bloomsbury	84-7888-561-7	Harry Potter and the Philosophers Stone
10.	184	Éditions Gallimard	987-1144-31-8	L Étranger

Figura 26: Resultado global

10. El resultado global se transforma a un formato admisible por OpenRefine (TSV, CSV, *SV, Excel (.xls y .xlsx), JSON, XML, RDF como XML o documentos de Google Data). En la implementación realizada el formato al

que se transforma el resultado global es CSV, por la simplicidad en la forma de representar una tabla.

11. OpenRefine toma el control realizando el flujo de trabajo básico para la limpieza, transformación y exportación de datos en grafos RDF.

    Una vez convertido el resultado global a un formato admisible por OpenRefine (CSV, de acuerdo a la implementación realizada), este toma el control y muestra un área de texto con la vista final integrada expresada en CSV. (Ver anexo 5, Figura 29)

    De esta manera, el usuario puede eliminar o modificar alguna de las columnas y/o filas. Seguidamente se presiona el botón “Next” y se muestra una tabla con la vista integrada final.

    Dentro de esta ventana, OpenRefine brinda al usuario la posibilidad de realizar ajustes para el paso de los datos hacia un nuevo proyecto, en el que será posible su exportación en varios formatos, incluidos los grafos RDF. (Ver anexo 5, Figura 30)

    Una vez concluidos los ajustes por parte del usuario, se procede a la creación de nuevo proyecto escribiendo un nombre para el mismo y presionando el botón “Create Project”.

    Cuando se crea un nuevo proyecto se muestra una ventana que contiene una nueva tabla con los datos, en la que es posible la limpieza, transformación y exportación de los mismos. (Ver anexo 5, Figura 31)

A partir de la aplicación del método teórico obtenido al caso de estudio se pudo constatar que el mismo resuelve el problema de integrar datos provenientes de múltiples bases de datos relacionales. Además, su implementación se incorpora satisfactoriamente a OpenRefine para posibilitar el trabajo de integrar los datos y seguidamente limpiar, transformar y exportar los resultados de la integración.

3.5. Conclusiones parciales

A partir de la aplicación de 60 casos de prueba de caja blanca, se detectaron 5 resultados no satisfactorios, los cuales fueron solucionados luego de dos iteraciones.

A partir de la aplicación de 30 casos de prueba de caja negra, se detectaron 12 resultados no satisfactorios, los cuales fueron solucionados luego de dos iteraciones.

A partir de la aplicación del método obtenido al caso de estudio se pudo constatar que se resuelve el problema de integrar datos provenientes de múltiples bases de datos relacionales en OpenRefine para la posterior limpieza, transformación y exportación de la vista integrada final en grafos RDF.

CONCLUSIONES GENERALES

A partir de la revisión bibliográfica realizada, no se identificó un método para la limpieza, transformación y exportación en grafos RDF de datos provenientes de la integración de múltiples bases de datos relacionales.

La consulta de la literatura científica permitió conocer que la herramienta OpenRefine brinda soporte para operaciones de limpieza, transformación y exportación en grafos RDF de datos, pero no permite la entrada de datos provenientes de bases de datos relacionales.

El desarrollo de una extensión para OpenRefine permitió la incorporación a la herramienta de un método para la integración de datos desde múltiples bases de datos relacionales.

A través de la aplicación del método propuesto a un caso de estudio se pudo constatar que la implementación de la extensión incorpora a OpenRefine la posibilidad de dar tratamiento a datos provenientes de múltiples bases de datos relacionales, permitiendo la generación de una vista integrada dinámica sobre la que se pueden realizar operaciones de limpieza y transformación.

RECOMENDACIONES

Las limitaciones del módulo DQP implementado están relacionadas con la amplia variedad de elementos que puede contener una consulta SQL. Actualmente, el DQP implementado soporta operaciones de integración basadas en consultas globales generadas a partir de criterios de igualdad entre columnas. Por tanto, se recomienda la mejora y optimización del mismo.

Debido a la complejidad de las operaciones para la integración, que incluyen la conformación del esquema de transformación y de los pares, la interfaz gráfica actual puede conllevar a confusiones al usuario. Por tal motivo, se recomienda mejorar el aspecto visual de la solución a partir de un análisis basado en criterios de usabilidad.

REFERENCIAS BIBLIOGRÁFICAS

1. HIDALGO, Y. y RODRIGUEZ, R. The semantic web: a brief overview. In: *Revista Cubana de Ciencias Informáticas*. 2013, Vol. 7, no. 1, pp. 76-85.

2. BERNERS-LEE, T. y HENDLER, J. The Semantic Web. In: *Scientific American Magazine*. 2001, pp. 29-37.

3. BERNERS-LEE, T. Linked Data. In: [online]. [Accessed 19 noviembre 2013]. Available from: http://www.w3.org/DesignIssues/LinkedData.html.

4. PRIYATNA, F. *RDF-based Access To Multiple Relational Data Sources*. Master. Madrid: Universidad Politécnica de Madrid, 2009.

5. CODD, E. A Relational Model of Data for Large Shared Data Banks. In: *Communications of the ACM*. 1970, Vol. 3, no. 6, pp. 377–387.

6. HEAT, T. y BIZER, CH. *Linked Data Evolving the Web into a Global Data Space* [online]. 1. S.l.: Morgan & Claypool Publishers, 2011. ISBN 9781608454310. Available from: www.morganclaypool.com.

7. KATSIS, Yannis y PAPAKONSTANTINOU, Yannis. View-based data integration. In: *Encyclopedia of Database Systems* [online]. S.l.: Springer, 2009. pp. 3332–3339. [Accessed 29 enero 2014]. Available from: http://link.springer.com/10.1007/978-0-387-39940-9_1072.

8. CALÌ, Andrea, CALVANESE, Diego y LENZERINI, Maurizio. Data Integration under Integrity Constraints. In: *Seminal Contributions to Information Systems Engineering. S.l.: s.n., 2013. pp. 335-352.*

9. DYCHÉ, Jill y LEVY, Evan. Customer Data Integration: Reaching a Single Version of the Truth. S.l.: John Wiley & Sons, 2011. ISBN 9781118046470.

10. VIANA, Iñaki Fernández de, HERNANDEZ, Inma, JIMÉNEZ, Patricia, RIVERO, Carlos R. y SLEIMAN, Hassan A. Integrating Deep-Web Information Sources. In: DEMAZEAU, Yves, DIGNUM, Frank, CORCHADO, Juan M., BAJO, Javier, CORCHUELO, Rafael, CORCHADO, Emilio, FERNÁNDEZ-RIVEROLA, Florentino, JULIÁN, Vicente J., PAWLEWSKI, Pawel y CAMPBELL, Andrew (eds.), *Trends in Practical Applications of Agents and Multiagent Systems* [online]. S.l.: Springer Berlin Heidelberg, 2010. Advances in Intelligent and Soft Computing, 71. pp. 311-320. [Accessed 29 enero 2014]. ISBN 978-3-642-12432-7, 978-3-642-12433-4. Available from: http://link.springer.com/chapter/10.1007/978-3-642-12433-4_37.

11. LENZERINI, Maurizio. A Tutorial on Data Integration. In: *DEIS'10 - Data Exchange, Integration, and Streaming*. Schloss Dagstuhl: s.n., 2010.

12. LENZERINI, Maurizio. Data Integration, Global scientic data infrastructures: The big data challenges. In: Capri: s.n., 2011.

13. MEDINA, Doris. *MODELO DE AUDITORÍA Y METADATOS PARA LA GESTIÓN DEL PROCESO DE INTEGRACIÓN DE DATOS* [online]. Pregrado. S.l.: Universidad de

las Ciencias Informáticas, 2012. [Accessed 29 enero 2014]. Available from: http://repositorio_institucional.uci.cu/jspui/jspui/handle/ident/4274.

14. WU, Tom. EII -ETL -EAI: What, Why, and How! In: *Information Integrator Advocate, IBM Software Group*. Taiwan. 2005.

15. RENHOLM, Kristoffer. *Enterprise Application Integration: An Investigative Case Study*. Maestría. Stockholm, Sweden: Royal Institute of Technology, 2011.

16. KOVANOVIC, Vitomir y DJURIC, Dragan. Highway: A Domain Specific Language for Enterprise Application Integration. In: *Proceedings of the 5th India Software Engineering Conference* [online]. New York, NY, USA: ACM, 2012. pp. 33–36. [Accessed 29 enero 2014]. Available from: http://doi.acm.org/10.1145/2134254.2134259.

17. FRANTZ, Rafael Z., REINA QUINTERO, Antonia M. y CORCHUELO, Rafael. A DOMAIN-SPECIFIC LANGUAGE TO DESIGN ENTERPRISE APPLICATION INTEGRATION SOLUTIONS. In: *International Journal of Cooperative Information Systems*. junio 2011, Vol. 20, no. 02, pp. 143-176. DOI 10.1142/S0218843011002225.

18. HERNÁNDEZ, Inma. *Enterprise Information Integration: An Unsupervised Proposal Forweb Page Classification*. Doctorado. Spain: University Of Sevilla, 2012.

19. KIMBALL, Ralph y CASERTA, Joe. The Data Warehouse. In: *ETL Toolkit. Practical Techniques for. Extracting, Cleaning,. Conforming, and. Delivering Data (Wisely Publication, Inc)* [online]. 1996, [Accessed 29 enero 2014]. Available from: https://sisis.rz.htw-berlin.de/inh2006/1235037.pdf.

20. HARTMAN, Yohanlena y RAMON, Dailen. *Implementacion del proceso de extraccion, transformacion y carga en un almacen de datos operacional para CIMEX* [online]. Pregrado. S.l.: Universidad de las Ciencias Informáticas, 2009. [Accessed 29 enero 2014]. Available from: http://repositorio_institucional.uci.cu/jspui/jspui/handle/ident/TD_2180_09.

21. RODRIGUEZ, Julio. *Modulo de extraccion, transformacion y replica de datos del Sistema para el Aseguramiento de la Actividad de Estructura y Composicion de las FAR version 3* [online]. Pregrado. S.l.: s.n., 2010. [Accessed 29 enero 2014]. Available from: http://repositorio_institucional.uci.cu/jspui/jspui/handle/ident/TD_03492_10.

22. LENZERINI, Maurizio. Data Integration: A Theoretical Perspective. In: *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* [online]. New York, NY, USA: ACM, 2002. pp. 233–246. Available from: http://doi.acm.org/10.1145/543613.543644.

23. ALPDEMIR, M., MUKHERJEE, A., GOUNARIS, A., PATON, N., WATSON, P., HERNANDES, A. y FITZGERALD, D. OGSA-DQP: A service for distributed querying on the grid. In: *Computer Science*. 2004, pp. 37.

24. LAWRENCE, R. A JDBC Driver Supporting Data Source Integration and Evolution. In: 2003, pp. 37-39.

25. VILALTA, J. *Introducción a UML*. S.l.: s.n., 2008.

26. MORERO, F. *Introducción a la POO*. S.l.: Grupo EIDOS, 2000.

27. FLANAGAN, D. *JavaScript: The Definitive Guide*. 6. S.l.: O'Reilly Media, 2011.

28. Marco de Trabajo. In: *Técnicas de ISW* [online]. 2010. [Accessed 4 abril 2014]. Available from: http://klcjw10.blogspot.com/2010/05/marco-de-trabajo.html.

29. LOUGHRAN, S. y HATCHER, E. *Ant in action* [online]. 2. S.l.: Manning Publications, 2007. ISBN 978-1932394801. Available from: http://www.manning.com/loughran.

30. CANOS, L. *Metodologías Ágiles en el Desarrollo de Software*. S.l.: s.n., 2011.

31. LETELIER, P. y PENADES, M. Metodologías ágiles para el desarrollo de software eXtreme Programming (XP). In: *Técnica administrativa*. 2006, Vol. 5, no. 26, pp. 1-17.

32. LARMAN, C. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. S.l.: Prentice Hall, 2004. ISBN 0131489062.

33. GAMMA, E. *Design Patterns: Elements of Reusable Object-Oriented Software*. S.l.: Addison Wesley, 1994. ISBN 0201633612.

34. PRESSMAN, R. *Ingeniería del software, un enfoque práctico*. Madrid: McGraw-Hill, 2002.

35. DEBEVOISE, N. *The MicroGuide to Process Modeling in BPMN*. S.l.: BookSurge Publishing, 2008. ISBN 978-1-4196-9310-6.

36. BECK, K. y WARD, C. A Laboratory For Teaching Object-Oriented Thinking. In: *OOPSLA'89 Conference Proceedings* [online]. New Orleans, Louisiana: ACM SIGPLAN Notices, 1989. Available from: http://c2.com/doc/oopsla89/paper.html#cards.

37. STARK, G. *Measurements to Manage Software Maintenance* [online]. S.l. The MITRE Corporation, 1997. [Accessed 3 junio 2014]. Available from: http://staff.unak.is/andy/MSc%20Software%20Maintenance2/Lectures/Measurements%20to%20Manage%20Software%20Maintenance%20-%20July%2097.htm.

38. RODRIGUEZ, M. y ORDOÑEZ, M. *La metodología XP aplicable al desarrollo del software educativo en Cuba* [online]. Havana: Universidad de las Ciencias Informáticas, 2007. Available from: http://repositorio_institucional.uci.cu/jspui/bitstream/ident/TD_0837_07/1/TD_0837_07.pdf.

39. BEIZER, B. *Software Testing Techniques*. 2. S.l.: Intl Thomson Computer Pr, 1990. ISBN 978-1850328803.