UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Modelos de indexación de informes estructurados DICOM-SR basados en grafos sobre infraestructuras Grid avanzadas TRENCADIS.

Curso

2012/2013

Trabajo Fin de Master

Máster Universitario en Computación Paralela y Distribuida

Autor

Lorenzo José Díaz de Haro

Director

Damián Segrelles Quilis

Codirector

Erik Torres Serrano

---

12 de septiembre de 2013

"La inspiración existe, pero tiene que encontrarte trabajando." - Pablo Picasso.

Agradecimientos.

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

En primer lugar quisiera dar las gracias a mi familia, especialmente a mis padres, Pepe y Luci, por apoyarme durante toda la vida, en los buenos y malos momentos, y por haberme enseñado que hay que luchar duro para llegar hasta aquí. Sin ellos esto jamás hubiera sido posible. Solo espero poder hacerlo tan bien como vosotros lo habéis hecho algún día.

A Belén, por entenderme y ayudarme desde hace tiempo. Por ser mi compañera y por tener la paciencia que tiene. Por leer este trabajo y corregir esos fallos que nunca ves. Te quiero.

A mi tutor, el Doctor Damián Segrelles, por confiar en mí, por sus orientaciones e ideas hacia donde caminar, por la visión que tiene sobre el proyecto, y por darme la oportunidad de trabajar en el GRyCAP. Por las horas dedicadas a leer y corregirme, le dedico un agradecimiento especial. Lo que comenzó por un pequeño paseo se está convirtiendo en una gran travesía.

Al Doctor Erik Torres, por sus aportaciones y sugerencias. Gracias por entender lo que quería decir y orientarme a como mejorarlo, y sobre todo por leer, corregir y traducir mis errores.

Al Doctor Ignacio Blanquer, por sus observaciones, su aporte de ideas, y su visión.

Mucha gente me queda por mencionar, pero todos tenéis que saber que os recuerdo y que os llevo conmigo allá donde vaya.

Resumen.

El continuo crecimiento exponencial de la información en los entornos empresariales comienza a suponer una amenaza para los analistas de datos. Es necesario disponer de nuevas capacidades, tecnologías y modelos que nos permitan obtener ventajas sobre los actuales, basados en herramientas tradicionales de gestión de datos.

Al igual que los entornos empresariales, los entes hospitalarios tienen el mismo problema, por lo que el volumen de información que generan, su velocidad de cambio y su variedad necesitan disponer de una serie de herramientas que permitan su tratamiento y visualización. Esto se enmarca en el ámbito Big Data, por lo que, es necesario solucionar la problemática expuesta siguiendo dicho término.

Por ello, este proyecto desarrolla una solución a la problemática existente con la indexación de los informes estructurados asociados a las imágenes DICOM, gestionados a través de infraestructuras avanzadas basadas en TRENCADIS, dentro del contexto de Big Data.

En esta tesis se ha desarrollado un modelo en el contexto Big Data basado en la teoría de grafos y se han diseñado e implementado los componentes necesarios para la integración del mismo en TRENCADIS, de forma que la indexación se realizará bajo un entorno Grid y con una base de datos noSQL basada en grafos.

Se presentan una serie de pruebas para comparar los modelos propuestos con los ya existentes, y, aunque en un primer modelo el resultado no es el esperado, al refinarlo y generar un segundo modelo los resultados obtenidos mejoran la implementación actual, dando pie a futuros trabajos de tratamiento y visualización de datos.

Índice de contenidos.

AGRADECIMIENTOS. **5**

RESUMEN. **7**

ÍNDICE DE CONTENIDOS. **9**

ÍNDICE DE ILUSTRACIONES. **11**

1. INTRODUCCIÓN. 13
   1.1. Visión general. *13*
   1.2. Motivación. *14*
   1.3. Organización de la tesis de master. *16*
2. ESTADO DEL ARTE. 17
   2.1. Tecnologías y estándares actuales. *17*
   2.1.1. Servicios GRID en Globus Toolkit 4.2. 17
   2.1.2. El *middleware* gLite. 19
   2.1.2.1. El servidor de catálogo AMGA. 20
   2.1.2.2. El servidor de catálogo LCG (ó LFC) 21
   2.1.3. El estándar DICOM-SR. 22
   2.1.4. La base de datos noSQL Neo4j. 24
   2.2. Estado actual de TRENCADIS. *24*
   2.2.1. La tecnología TRENCADIS. 25
   2.2.1.1. CORE Services. 26
   2.2.1.2. SERVER Services. 29
   2.2.2. Despliegue de TRENCADIS en la nube. 30
3. OBJETIVOS. 33
4. DESARROLLO DEL PROYECTO. 35
   4.1. Modelo V.1.0 : Representación completa de las estructuras de los informes estructurados y de las instancias de los mismos en un grafo. *35*
   4.1.1. Estudio, análisis y diseño del modelo versión 1.0. 35
   4.1.1.1. Representación de informes estructurados basados en DICOM-SR mediante el estándar XML 35
   4.1.1.2. Consideraciones previas de los grafos. 39
   4.1.1.3. Análisis y diseño del modelo 1.0 51
   4.1.1.4. Ventajas del modelo 1.0. 61
   4.1.2. Desarrollo e implementación. 63
   4.1.2.1. Descripción de los módulos. 63
   4.1.2.2. Descripción de los servicios. 65
   4.1.3. Pruebas realizadas sobre el modelo 1.0. 67
   4.1.4. Conclusiones. 70
   4.2. Modelo V.2.0: Representación simplificada de las estructuras de los informes estructurados y de las instancias de las mismas en un grafo *71*
   4.2.1. Estudio, análisis y diseño del modelo versión 2.0. 71
   4.2.1.1. Creación de cabecera de datos de la instancia. 74
   4.2.1.2. Asignación de valores a los nodos identificadores. 75
   4.2.1.3. Características de flexibilidad y reutilización de nodos en el modelo. 79
   4.2.1.4. Relación entre las distintas ontologías. 80
   4.2.2. Desarrollo e implementación. 80
   4.2.2.1. Descripción de los módulos. 80
   4.2.2.2. Descripción de los servicios. 86
   4.2.3. Pruebas realizadas sobre el modelo 2.0. 87
   4.2.3.1. Pruebas similares al modelo 1.0. 87
   4.2.3.2. Pruebas de carga de datos al modelo 2.0. 88
   4.2.4. Conclusiones del modelo 2.0. 89

5. CONCLUSIONES Y APORTACIONES. 91
6. ARTÍCULOS ASOCIADOS. 93
7. TRABAJOS FUTUROS 95

BIBLIOGRAFÍA. 97

Índice de ilustraciones.

Figura 1 - Convergencia entre tecnologías [19]. 18
Figura 2 - Arquitectura de los servicios Grid [23]. 19
Figura 3 - Arquitectura de componentes de gLite [25]. 20
Figura 4 - Nombres de ficheros en LCG. 22
Figura 5 - Modelo de datos de DICOM. 23
Figura 6 – Ejemplo de DICOM-SR extraído de [2]. 23
Figura 7 - Despliegue actual de TRENCADIS. 25
Figura 8 - Infraestructura de TRENCADIS. 26
Figura 9 -DICOM Storage Service. 28
Figura 10 - Key Server Service. 29
Figura 11 - Estructura del DICOM-SR. 36
Figura 12 – Contenido de *Content Item* [34]. 37
Figura 13 - Estructura del módulo de contenido [34]. 37
Figura 14 - Tipo TEXT. 38
Figura 15 - Grafo inicial no dirigido. 40
Figura 16 - Segundo grafo para realizar la fusión. 41
Figura 17 - Grafo resultado de la operación de fusión. 41
Figura 18 - Camino 1 entre el nodo 5 y el nodo 8. 42
Figura 19 – Camino 2 entre el nodo 5 y el nodo 8. 42
Figura 20 – Primera arista con id=[3,8]. 43
Figura 21 - Segunda arista con id=[3,8]. 43
Figura 22 – Camino 1 que incluye al nodo 7. 44
Figura 23 - Camino 2 que incluye al nodo 7. 44
Figura 24 - Identificación única de las aristas. 45
Figura 25 - Grafo con propiedad path conteniendo aristas. 46
Figura 26 - Camino correcto desde el nodo 5 al nodo 8 a través de la propiedad *path*. 47
Figura 27 - Aristas con *path* almacenando los identificadores de los nodos. 47
Figura 28 – Ejecución de consulta en *Cypher*. 48
Figura 29 - Ejecución de consulta en *Cypher*. 48
Figura 30 - Ejecución de consulta en *Cypher*. 49
Figura 31 - Ejecución de consulta en *Cypher*. 49
Figura 32 - Ejemplo de grafo con problema. 50
Figura 33 - Fusión de grafos de las figuras 32 y 16. 50
Figura 34 - Grafo final tras usar el grado de multiplicidad. 51
Figura 35 - Identificación de un nodo. 52
Figura 36 - Representación XML de las jerarquías existentes. 53
Figura 37 - Representación gráfica de las jerarquías existentes. 53
Figura 38 - Alta de nueva estructura.. 54
Figura 39 - Representación gráfica de ambas estructuras reutilizando nodos. 54
Figura 40 - Representación final del árbol de jerarquías. 55
Figura 41 - El problema de los múltiples elementos repetidos. 56

Figura 42 - Reutilización de nodos sin identificador de multiplicidad. 57
Figura 43 - Reutilización de nodos con identificador de multiplicidad. 57
Figura 44 - Grafo con los 3 niveles de jerarquía aplicando el grado de multiplicidad. 58
Figura 45 – Fragmento de fichero XML de una instancia de una exploración mamaria. 60
Figura 46 – Grafo de representación de cabecera de instancia de ontología. 60
Figura 47 – Representación de valor de un elemento en el grafo. 61
Figura 48 – Ejemplos de grafos para fusionar. 62
Figura 49 – Grafos de la figura 48 fusionados. 63
Figura 50 - Grafo simplificado de la cabecera de la ontología. 72
Figura 51 - Representación de la mama derecha y la mama izquierda dentro de la ontología. 73
Figura 52 – Fragmento de representación de masa situado en la mama derecha y en la izquierda. 74
Figura 53 – Estructura de la cabecera de la instancia. 74
Figura 54 – Dependencia entre los nodos de instancias. 75
Figura 55 – Fragmento de instancia de informe estructurado. 75
Figura 56 – Representación del identificador del informe estructurado para la figura 55. 76
Figura 57 – Representación de identificador de masa en la mama derecha del informe estructurado de la figura 55 y figura 56. 77
Figura 58 – Representación de valores de la masa del informe estructurado de la figura 55 y figura 56. 78
Figura 59 –Representación de valores de los hallazgos asociados del informe estructurado de la figura 55 y figura 56. 78
Figura 60 – Representación de la cabecera de una instancia en la ontología correspondiente. 83

1. Introducción.

1.1. Visión general.

La medicina moderna, en general, y en las áreas relacionadas con la imagen médica, en particular, no puede concebirse sin información en formato digital: radiografías, resonancias magnéticas, ecografías, informes etc. Con ellas se encuentra fuertemente ligada la computación, con el fin de abordar las distintas situaciones que se presentan, ya sea, tanto para almacenar, como para procesar toda esta información.

En este sentido, el uso de estándares para los formatos digitales y de computadores para su proceso proporcionan grandes ventajas, como la posibilidad de aplicar algoritmos avanzados, facilidad al acceso y organización de los mismos, colaboración entre distintas entidades, etc. Concretamente, en los hospitales de todos los países desarrollados del mundo se utiliza *Digital Imaging and Communication in Medicine* (DICOM) [1] como estándar para el almacenamiento de imágenes médicas y comunicación de los dispositivos que los gestionan. Este estándar tiene una extensión que permite estructurar la información asociada a las imágenes, posibilitando la creación de informes estructurados que pueden ser manejados por los propios dispositivos empleados en los hospitales. Dicha extensión corresponde al *DICOM Structured Reporting* (DICOM-SR) [2].

En la actualidad, es inmenso el volumen de datos generado, relacionado con la imagen digital a través del estándar DICOM y la información asociada a esta, a través de los informes estructurados DICOM-SR, procedentes de los centros hospitalarios. Por ello, es preciso disponer de herramientas de tratamiento y visualización de datos que nos permitan obtener resultados en un tiempo razonable.

La computación Grid solucionó, en gran medida, la problemática de organizar la información a nivel inter-hospitalario, permitiendo la creación de entornos colaborativos, a través de Organizaciones Virtuales (VO) [3] que permitían compartir recursos de almacén y de cómputo.

Las necesidades actuales han cambiado, dado que la cantidad de información que se puede almacenar en este tipo de entornos colaborativos es inmensa, de diferente origen y formato y sin una estructura predeterminada. Más si cabe, si hablamos de los informes médicos DICOM-SR, dentro de los cuales se puede incluir una gran variedad de terminologías y léxicos estándares, como son RadLex [4], ICD-10 [5] o SNOMED-CD [6]. Todo ello, hace necesario acudir a nuevos modelos de representación correspondiente a los datos incluidos en los informes estructurados, que nos permitan seguir obteniendo resultados razonables, y más aún, en un marco de información desestructurada en su conjunto como es el actual. Con ello se consigue tender puentes entre los diferentes formatos de datos para posibilitar la correlación de la información,

con el fin de lograr una extracción del conocimiento más eficiente.

Big Data [7] se presenta como un marco de solución a la necesidad planteada, y con ello, las distintas herramientas noSQL [8] que existen a su alrededor. Estas herramientas integradas en infraestructuras Grid avanzadas como TRENCADIS [9], nos ofrecerían un conjunto de soluciones óptimas para el problema del almacenamiento y de la representación de la información. Este es, precisamente, el marco principal de nuestro trabajo, donde el tratamiento y la visualización de los datos será el camino abierto a posteriores estudios.

1.2. Motivación.

La motivación de esta tesis se fundamenta en mejorar la indexación y búsqueda de información en informes estructurados médicos, fundamentados en el estándar DICOM-SR y asociados a imágenes DICOM, en infraestructuras Grid avanzadas basadas en TRENCADIS, todo esto en un contexto de Big Data.

Existen diversos puntos que justifican la motivación de esta tesis:

En primer lugar, la representación de un informe DICOM-SR corresponde con un grafo con estructura de árbol. La aparición de tecnologías noSQL basadas en grafos, que dan soporte a contextos de Big Data, parecen ser idóneas para representar los informes DICOM-SR, dado que están formados por nodos, los cuales contienen propiedades que los describen y aristas que los unen. De esta manera, se forma un conjunto de relaciones que permite modelar cualquier escenario posible respecto a informes estructurados, independientemente de su tamaño y estructura.

En segundo lugar, los datos generados en los informes estructurados de los distintos departamentos de los centros médicos tienen diversas procedencias: imágenes radiológicas, documentos escaneados acerca de pruebas o pacientes, informes médicos, grabaciones de voz o video acerca de una operación, simulaciones, modelos tridimensionales, etc. Todo esto obliga a disponer de capacidad de almacenar, consolidar y fusionar información de los informes estructurados de forma óptima, ya que dichos datos, aunque de forma individual en cada informe tengan una estructura propia bien definida, en su conjunto se encuentran desestructurados y poco o nada relacionados entre sí. Además, es necesario adaptarse a los nuevos formatos o plantillas de informes que van apareciendo, y que cada vez son más sofisticados (por ejemplo un video en alta definición).

En tercer lugar, integrar un modelo de grafos para la indexación de informes DICOM-SR en una infraestructura Grid avanzada TRENCADIS, el cual nos permitiría almacenar, manipular y relacionar, de forma sencilla, una gran cantidad de datos de estructura diferente y de origen diverso, ya que, en este tipo de modelo basado en grafo, todo surge de las relaciones que se producen dentro del mismo, a través de las aristas. En TRENCADIS la información se distribuye entre diferentes centros de una forma transparente para los usuarios y coordinada a través de VOs creadas por aquellos. Por ello, se requería de un nuevo componente en TRENCADIS que permitiera la creación y

modificación de metamodelos basados en grafos, capaces de representar los informes DICOM-SR, así como sus instancias y las posibles relaciones que se presentaran entre ellas. En el supuesto de un gran despliegue de TRENCADIS dentro de un contexto de Big Data, con un funcionamiento de 24 horas al día, 365 días al año, podemos encontrarnos con un volumen de datos enorme, generado diariamente, en el que la tecnología de bases de datos utilizada en TRENCADIS (actualmente AMGA [32]) sea insuficiente para manejarlo de forma eficaz.

Y en cuarto lugar, entendemos que este trabajo se enmarca en el contexto de Big Data, dado que el reto es gestionar ingentes cantidades de información producidas por distintos centros hospitalarios organizados a través de VOs, con formatos y estructuras diferentes, relacionando los datos y permitiendo su consulta de una forma rápida y ágil. Todo esto acarrea una dificultad a la hora de trabajar con bases de datos convencionales, ya que resulta muy difícil manejar estos conjuntos de datos, debido a su gran tamaño y a la desconexión entre sí que presentan. Asimismo, existe una necesidad de usar e interpretar los grandes conjuntos de datos, por lo que, es preciso, entender las tres características esenciales en un contexto de Big Data: volumen, variedad y velocidad.

- Volumen: Un error común es afirmar que es la característica esencial desde el punto de vista de la utilidad. El continuo crecimiento de los datos es una cuestión fundamental, pero sin duda en la variedad de los datos, y en la vida útil de los mismos (definido como "velocidad") es donde podemos encontrar el verdadero valor añadido.
- Variedad: Describe el número de tipos diferentes de datos que podemos encontrar, interpretar y analizar a la vez.
- Velocidad: Hace referencia a la vida útil de los datos, ya que no tiene mucho sentido almacenar datos desactualizados.

Actualmente se habla de una cuarta característica dentro de Big Data: veracidad. Los datos han de ser seguros, verdaderos y disponer de un cierto nivel de confidencialidad.

Esta tesis parte de un trabajo ya realizado con anterioridad en TRENCADIS, donde la indexación y búsqueda de informes DICOM-SR se realiza sobre un catálogo de objetos implementado sobre componentes de una infraestructura Grid avanzada, concretamente los componentes AMGA, LFC y SE del *middleware* gLite [12], pero no basado en el contexto de Big Data.

Además, no solo se busca indexar la información, sino también, en un futuro, sentar las bases para poder realizar *data mining* o extracción de datos, de forma que se pueda registrar hábitos de los pacientes con una determinada enfermedad, analizar evoluciones de enfermedades y asignar medicación en consonancia, etc. Y por supuesto la creación futura de una serie de "*rankings*" que nos facilitarán el diagnóstico avanzado de enfermedades.

1.3. Organización de la tesis de master.

La tesis de master se encuentra estructurada en diversos capítulos, de forma que tiene el siguiente esquema:

- Capítulo 1: Visión general de la tesis, y motivación de la misma.
- Capítulo 2: Imagen de la situación actual de las tecnologías usadas durante el desarrollo del proyecto.
- Capítulo 3: Objetivos de la tesis.
- Capítulo 4: Descripción del trabajo realizado, desarrollando los componentes necesarios para crear el nuevo servicio de catálogo basado en un modelo de grafos e implementado con Neo4j [13].
- Capítulo 5: Conclusiones obtenidas en el proyecto.
- Capítulo 6: Breve descripción de los artículos publicados.
- Capítulo 7: Mejoras futuras sobre el trabajo realizado.

2. Estado del arte.

2.1. Tecnologías y estándares actuales.

A continuación se presenta el conjunto de tecnologías y estándares utilizados en esta tesis de master. Además, se presentará el estado actual del *middleware* TRENCADIS, a partir del cual se ha realizado este trabajo.

2.1.1. Servicios GRID en Globus Toolkit 4.2.

Antes de hablar de Globus Toolkit [14] es necesario abordar la arquitectura OGSA [15], de Servicios Grid [16], y de las especificaciones WSRF [17].

La arquitectura OGSA (*Open Grid Services Architecture*) fue estandarizada por GGF (*Global Grid Forum*) como un *framework* con un amplio uso para la integración de Sistemas Distribuidos. OGSA propuso definir el alcance de los servicios que son necesarios para las aplicaciones de Grid, intentando estandarizar una arquitectura.

Con ello propuso una implementación de la misma denominada OGSI (*Open Grid Services Infrastructure*) [16]. El fin era definir mecanismos para crear, gestionar e intercambiar información entre entidades llamadas Servicios Grid, utilizando tecnología de Grid y Servicios Web. Dicha implementación fue usada en Globus Toolkit 3, pero no en Globus Toolkit 4.

Debido a que OGSA es una especificación a la cual podría adaptarse cualquier *middleware* distribuido (CORBA, RMI, Servicios Web, etc.) los Servicios Web son los que mejor se adaptan. De esta forma los Servicios Grid aparecen como una ampliación de los servicios Web, siendo su arquitectura especificada por el OGSA. Los Servicios Grid superaban las limitaciones que tenían los Servicios Web:

- Los Servicios Web no mantienen el estado de una invocación a otra, en contra de los Servicios Grid, que sí lo mantienen.
- No es posible crear varias instancias del mismo Servicio Web y destruirlos, según sean necesarios, en contra de los Servicios Grid que sí que pueden.
- Los Servicios Web no incluyen otros servicios como notificaciones persistencia, gestión de ciclo de vida, etc.

Las especificaciones WSRF (*Web Service Resource Framework*) son una serie de especificaciones sobre Servicios Web publicadas por OASIS [18], con la finalidad de obtener Servicios Web con estado, de forma que los clientes de los servicios se comunican con servicios WSRF, los cuales representan recursos, y permiten realizar operaciones sobre los mismos. Al final se produce una convergencia entre ambas

tecnologías.

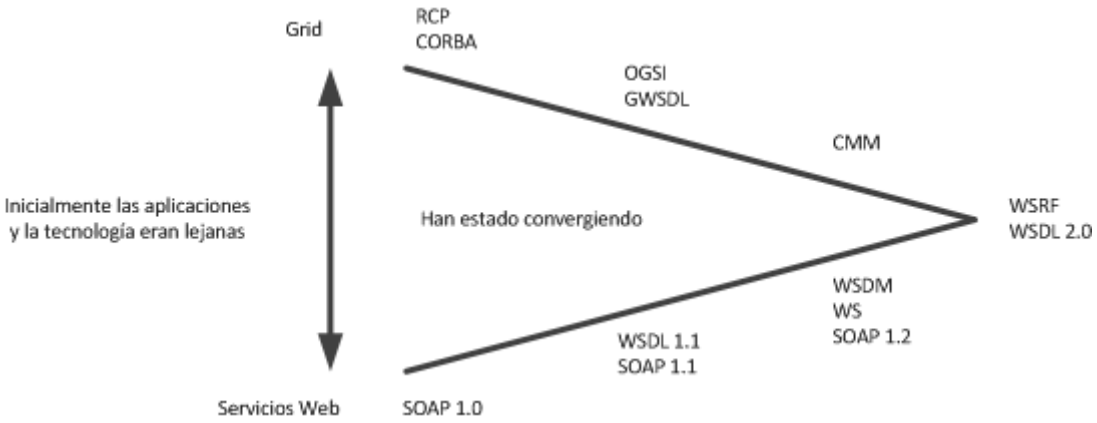

Figura 1 - Convergencia entre tecnologías [19].

Con todo ello Globus Alliance desarrolló un *middleware open source* denominado Globus Toolkit, siendo el más usado para la construcción de Grids. Se define como una "suma de servicios", de forma que está compuesta de múltiples servicios que implementan distintos componentes necesarios para el Grid. Se puede comprobar en Globus Toolkit que:

- Provee de librerías y componentes que permiten el desarrollo de aplicaciones Grid orientadas a servicios.
- El núcleo de Globus abarca características básicas relacionadas con seguridad, acceso, administración de recursos, movimiento y administración de datos, descubrimiento de recursos, etc.

Globus Toolkit 4.2 dispone de una implementación completa de WSRF y gran parte de OGSA.

Con todo esto Globus Toolkit 4.2 proporciona una serie de servicios, de forma que los expone en forma de operaciones a realizar, con una sintaxis que debe ser utilizada. Con todo ello, la implementación de los servicios es independiente, pudiendo encontrarse en diferentes lenguajes de programación respecto a la aplicación cliente (incluso pudiendo ejecutarse en plataformas distintas el cliente y el servidor).

La invocación de un Servicio Grid se produce a través de un punto de entrada denominado URI, donde dicha información viene proporcionada por el Servicio de Información ó *Monitoring and Discovery System* [20] (MDS), y de un *Web Services Description Language* [21] (WDSL) que indica la forma en que se invoca el Servicio Web. A continuación una petición bajo el protocolo de alto nivel *Simple Object Access Protocol* [22] (SOAP), permite la ejecución de dicho Servicio Web.

En la figura 2 podemos ver un diagrama de cómo se produce una invocación de forma correcta de un Servicio Grid.

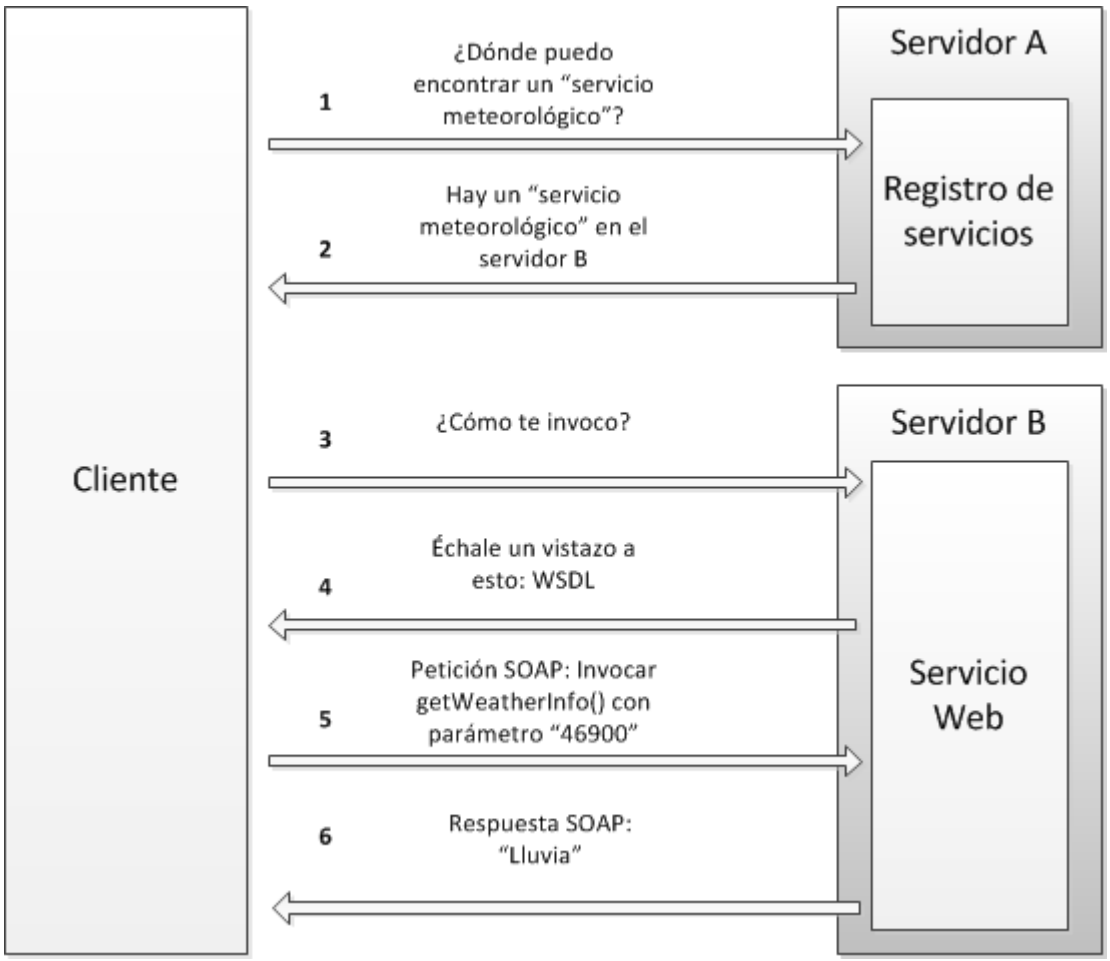

Figura 2 - Arquitectura de los servicios Grid [23].

2.1.2. El *middleware* gLite.

Se trata de un *middleware* ligero para la computación Grid [24], basado en una arquitectura orientada a servicios (SOA). Surge a partir del esfuerzo de diversas instituciones de investigación académicas e industriales, a raíz del proyecto EGEE.

Está formado por un conjunto integrado de componentes que proporcionan una serie de servicios, de forma que es posible construir la infraestructura de un sistema Grid usando dichos componentes.

Entre los servicios que proporciona podemos hablar de:

- Servicios de Acceso.
- Seguridad.
- Sistema de información.
- Sistema de Gestión de Trabajos.
- Sistema de Gestión de Datos.

En la figura 3 podemos visualizar la arquitectura de componentes que presenta gLite, sobre la cual trabajaremos con los Servicios de Gestión de Datos, más concretamente sobre el Catálogo de Metadatos.

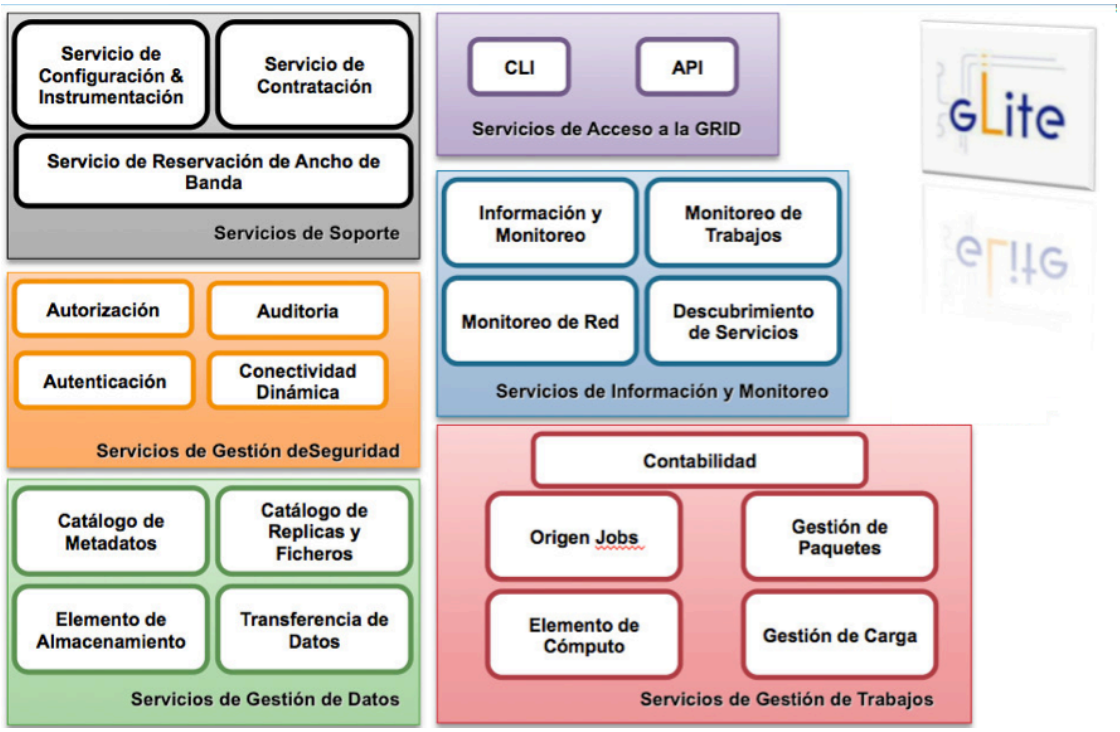

Figura 3 - Arquitectura de componentes de gLite [25].

El Sistema de Metadatos (AMGA) permite la extracción y el almacenamiento transparentes de los metadatos de los ficheros, trabajos o cualquier información. El *Storage Element* (SE) y el *File Catalogue* (LFC) [26] permiten gestionar el almacenamiento y transferencia de archivos grandes o facilitar la disponibilidad y localización de archivos de datos requeridos por los *jobs*.

En este trabajo de master se persigue modificar el sistema de Metadatos con un sistema propio basado en un modelo de grafos sobre la base de datos Neo4j, e integrándolo con los demás componentes (SE y LFC).

2.1.2.1. El servidor de catálogo AMGA.

AMGA es un servidor de metadatos que proporciona una interface sobre una base de datos relacional, convirtiéndose de esta forma en el Catálogo de Metadatos de gLite.

Permite a los usuarios adjuntar información de metadatos a los archivos almacenados en la red, donde los metadatos pueden ser cualquier dato relacional almacenado en un sistema de bases de datos relacionales (RDBMS). Además, los metadatos en AMGA también se pueden almacenar de forma independiente de los archivos asociados, lo que permite que AMGA pueda ser utilizado como una herramienta de acceso general a bases de datos relacionales en el Grid.

En este sentido, al trabajar con un catálogo de metadatos, utilizamos una serie de conceptos nuevos para referirnos al modelo. Esto es debido a que AMGA estructura los datos en directorios y ficheros (o sea en forma de árbol). Los conceptos a destacar en dicha estructura son:

- Colección: Conjunto de esquemas que representan un catálogo de metadatos.
- Esquema: Un conjunto de atributos. En el modelo relacional serían las tablas.
- Atributos: Par clave/valor con un tipo de información. En el modelo relacional serían las columnas, y en este caso vienen representados como directorios.
- Entradas: Viven en un esquema y asigna valores a los atributos. En el modelo relacional serían las filas, y en este caso vienen representados como ficheros dentro de los directorios.

A parte de esto, AMGA implementa una serie de mecanismos de autorización con listas de control de acceso y permisos de fichero tipo UNIX, por lo que proporciona un nivel de seguridad adecuado en un entorno Grid heterogéneo.

Dispone de soporte de transacciones, e implementa muchas de las funcionalidades del estándar SQL (de hecho ha de funcionar sobre una base de datos relacional).

2.1.2.2. El servidor de catálogo LCG (ó LFC).

Los datos de las aplicaciones GRID requieren ser almacenados en ficheros, y estos a su vez se almacenan en elementos de almacenamiento (*Storage Elements* ó SE). Además es posible generar réplicas de los ficheros en diferentes recursos.

Por ello necesitamos un componente para poder localizar los ficheros de datos (y sus réplicas) de forma transparente. Este componente es el LFC (*LCG File Catalogue*), que mantiene asociados el nombre lógico del fichero a las direcciones físicas donde se encuentran.

Existen 4 formas distintas de poder acceder a un fichero mediante el LFC [4]: GUID (*Grid Unique IDentifier*), LFN (*Logical File Name*), SURL (*Storage URL*) y TURL (*Transport URL*):

- GUID: Cualquier fichero puede ser identificado por su GUID, que es asignado en el momento del registro del fichero, y sigue el estándar UUID para garantizar que se trata de un identificador único. Dicho GUID tiene el formato `guid:<identificador único>`. Todas las réplicas de un archivo compartirán el mismo GUID.
- LFN: Mecanismo más común por el que el usuario accede al fichero, ya que se trata de un nombre lógico del fichero (un alias) asignado por el usuario. El formato de uso es muy similar al del GUID, siendo `lfn:<cualquier alias>`.
- SURL: Proporciona información sobre la ubicación física del archivo (nombre de host y ruta) en un SE. Tiene el siguiente formato (dependiendo del tipo de SE que contenga el fichero):

- `sfn:<SE_hostname>/<local_string>`
- `srm:<SE_hostname>/<local_string>`.

- TURL: Se trata de la información necesaria para recuperar una réplica, indicando el protocolo (incluso el puerto) de acceso soportado por el SE. También se ha de indicar el host y la ruta de acceso.

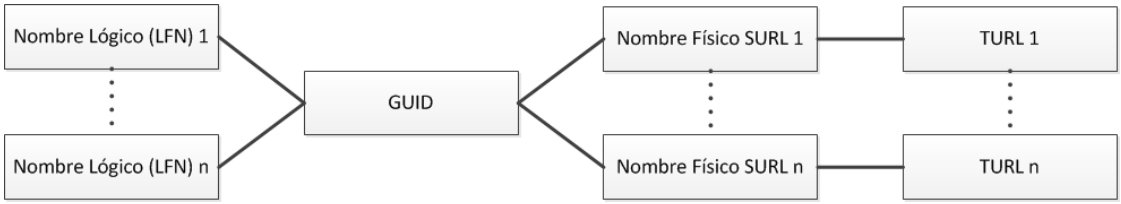

Figura 4 - Nombres de ficheros en LCG.

Mientras que los GUID y LFN hacen referencia a los archivos y no a las réplicas, además de no decir nada acerca de las ubicaciones, los SURL y TURL proporcionan información acerca de dónde se encuentra una réplica física.

2.1.3. El estándar DICOM-SR.

El estándar DICOM (*Digital Imaging and Communications in Medicine*) es un estándar ubicuo en la industria de la imagen de cardiología y radiología, habiéndose extendido a otros campos de la medicina, con el fin de ayudar al manejo, almacenamiento, impresión y transmisión de dichas imágenes, por lo que para ello define como ha de ser el formato de fichero, así como el protocolo de comunicación, permitiendo la integración de múltiples dispositivos dentro del mismo sistema de almacenamiento y comunicación [27] [28].

El modelo de datos define las entidades de información que todo objeto DICOM ha de tener. Este está compuesto de una cabecera del paciente, el estudio al que hace referencia, la serie a la que pertenece, y las imágenes (figura 5).

También se ha desarrollado los estándares necesarios para los documentos que incorporan referencias a las imágenes, así como sus datos asociados. A dicho desarrollo se le denomina DICOM-SR (*DICOM Structured Reporting*) [2] [29]. DICOM-SR es un documento estructurado que contiene texto con enlaces a otros datos como imágenes, formas de onda y coordenadas espaciales o temporales.

Figura 5 - Modelo de datos de DICOM.

Como cualquier otro objeto DICOM, DICOM-SR dispone de una cabecera y de contenido [30]. La cabecera es la misma que se ha descrito con anterioridad, solo que en vez de imágenes podemos encontrar información jerarquizada en distintos niveles, donde los elementos de información son de distintos tipos, como texto, valores numéricos, coordenadas espaciales o temporales y las referencias a las imágenes o formas de onda. Los elementos de información se encuentran conectados jerárquicamente en forma de árbol.

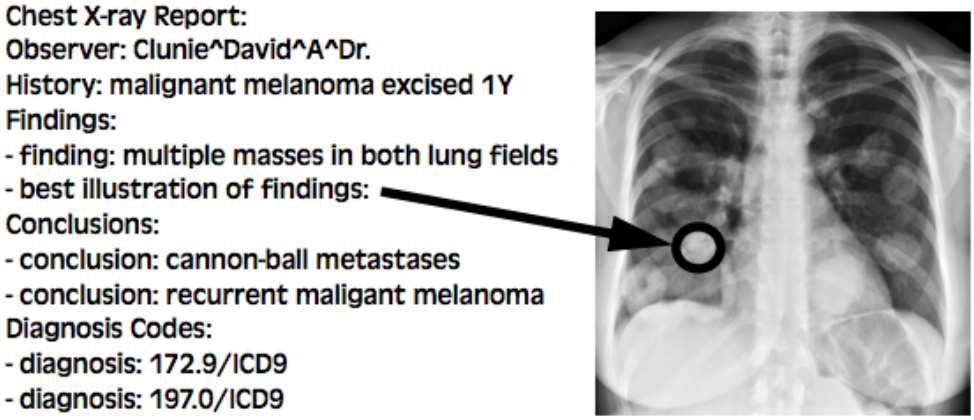

Figura 6 – Ejemplo de DICOM-SR extraído de [2].

El estándar ha adoptado el lenguaje XML para codificar el documento estructurado. Esto es debido a que DICOM-SR puede ser transformado de forma muy fácil a otro formato de documento.

2.1.4. La base de datos noSQL Neo4j.

En el momento actual ha aparecido un nuevo paradigma para la representación de datos, denominado noSQL, el cual consiste en usar bases de datos que no siguen el modelo relacional tradicional, por lo que no hacen uso del lenguaje SQL.

De forma genérica, una gran mayoría de estas bases de datos suelen funcionar como un gran mapa, donde es posible poner cualquier tipo de datos y recuperarlos. Estas no suelen tener un esquema, por lo que evolucionan y se adaptan con una gran rapidez. Además evitan la operación de *join* y suelen escalarse horizontalmente [30].

Este tipo de bases de datos nace a partir de la necesidad de procesar grandes cantidades de datos en un tiempo reducido. Las bases de datos relacionales tradicionales no se ajustaban a dicha situación, por lo que era necesario un nuevo tipo que permitiera operaciones de inserción y recuperación de forma rápida, perdiendo flexibilidad en tiempo de ejecución, pero con ganancias significativas en escalabilidad y rendimiento.

Este sistema gestor de bases de datos encaja en el término Big Data, principalmente por su capacidad de almacenar ingentes cantidades de información desestructurada, además de ajustarse al paradigma noSQL.

Estas bases de datos se suelen clasificar según su forma de almacenar los datos y comprenden categorías como clave-valor, implementaciones de *BigTable*, bases de datos documentales y bases de datos orientadas a grafos. A este último grupo pertenece Neo4j.

Neo4j es una base de datos orientada a grafos que se puede encajar perfectamente en el término "*Big data*". Algunas de las características de la base de datos son:

- Escalabilidad.
- Alto rendimiento.
- Lenguaje de consultas "*human friendly*".

Neo4j se compone de nodos y relaciones, ambos con posibilidad de tener propiedades. Las relaciones unen los nodos, formando caminos, con la posibilidad de realizar consultas de tipo transversal, que serían los caminos que unen los distintos nodos. Además de esto, es posible realizar búsquedas directamente sobre sus nodos o relaciones a través de sus propiedades, creando índices sobre ellos.

2.2. Estado actual de TRENCADIS.

TRENCADIS despliega una infraestructura Grid para almacenar, recuperar y procesar grandes cantidades de imágenes médicas y datos asociados a las mismas,

siendo almacenadas, tanto las imágenes como los datos asociados, en objetos DICOM [31].

El sistema permite buscar y manipular imágenes e informes estructurados, con el fin de relacionar los hallazgos clínicos, y poder diagnosticar y tratar diversas enfermedades, todo ello sobre un entorno Grid.

Actualmente existe una implementación real del sistema, con un prototipo de aplicación Web en el Hospital Dr. Peset de Valencia. Dicha aplicación permite a los radiólogos hacer uso de informes de diagnóstico de exploraciones de mamografía acerca del cáncer de mama. El despliegue actual sigue el esquema de la siguiente imagen:

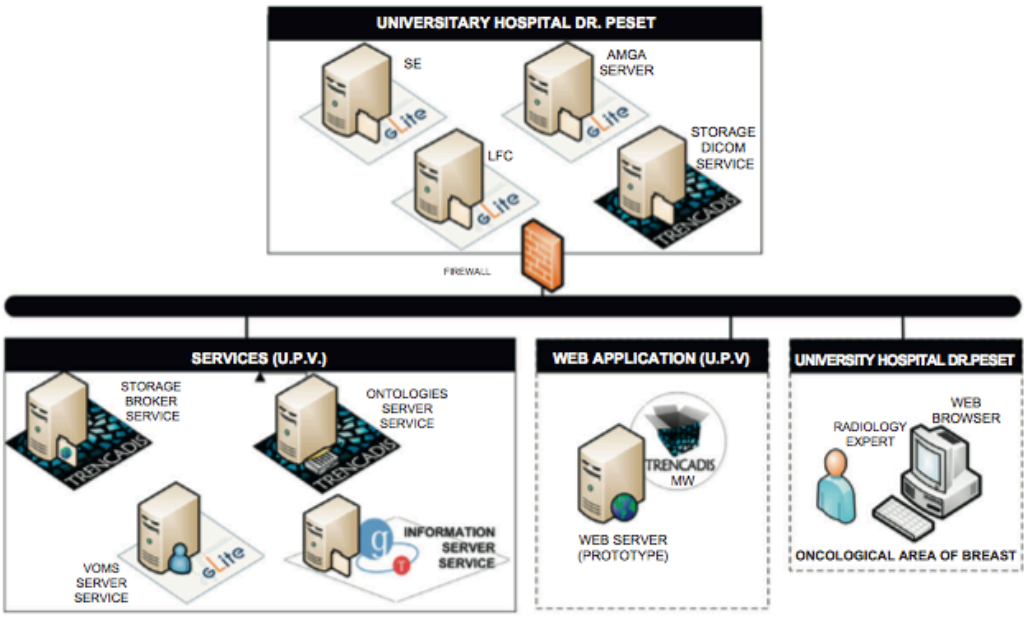

Figura 7 - Despliegue actual de TRENCADIS.

2.2.1. La tecnología TRENCADIS.

Antes de vislumbrar la tecnología que envuelve al *middleware* TRENCADIS, fue necesario definir una serie de puntos:

1. Los servicios de infraestructura necesaria para almacenar e indexar la información DICOM en los servicios Grid que TRENCADIS ofrece como recursos.
2. Cómo distribuir los recursos entre las distintas instituciones implicadas, permitiendo la posibilidad de incorporar nuevas instituciones médicas.
3. Fue necesario identificar los casos de uso más importantes que pueden participar en el prototipo propuesto y definir el flujo de información entre los servicios que intervienen en el mismo.
4. Se codificó y desplegó las plantillas DICOM-SR necesarias asociadas a los procesos clínicos pertenecientes al cáncer de mama y su seguimiento.
5. Se necesitó mejorar la expresividad de los servicios Grid desplegados.

Todo ello llevó a definir una arquitectura orientada a servicios (SOA), en la que la gestión de los recursos es llevada a cabo por distintos servicios de red, que siguen la especificación de arquitectura abierta (OGSA).

El despliegue de TRENCADIS, con todos sus servicios, en el hospital de referencia, se puede ver en la figura 7. Observamos en la misma figura que TRENCADIS se encuentra estructurado en varias capas:

- La capa de infraestructura proporciona los recursos básicos del entorno, que se ofrecen como interfaces estándares definidas en *Web Services Definition Language* (WSDL). Estos recursos utilizan protocolos como SOAP, LFC y AMGA, y formatos de datos tipo XML. Todo esto proporciona a las capas superiores una interfaz única para todos los recursos del mismo tipo.
- La capa de comunicación, la cual define los estándares y protocolos utilizados en la comunicación entre los distintos componentes del despliegue, dependen de los *BackEnds* (LFC o GridFTP en este caso). GridFTP se utiliza para la transferencia de grandes cantidades de datos, SOAP en la parte superior de HTTPS, donde se utiliza para la interacción de servicios de red y *Secure Socket Layer* (SSL) que se emplea para proteger las transmisiones con *BackEnds* AMGA.
- La capa de usuario se trata de una aplicación web que interacciona el *middleware* TRENCADIS, proporcionando acceso a los distintos objetos DICOM.

Los servicios de infraestructura de TRENCADIS vienen definidos en la capa de infraestructura y dentro de ellos podemos definir dos tipos: servicios del núcleo (*CORE Services*) y servicios de servidor (*SERVER Services*). Vamos a constatar que en ambos casos es recomendable que el sistema funcione bajo el S.O. Scientific Linux 6.7.

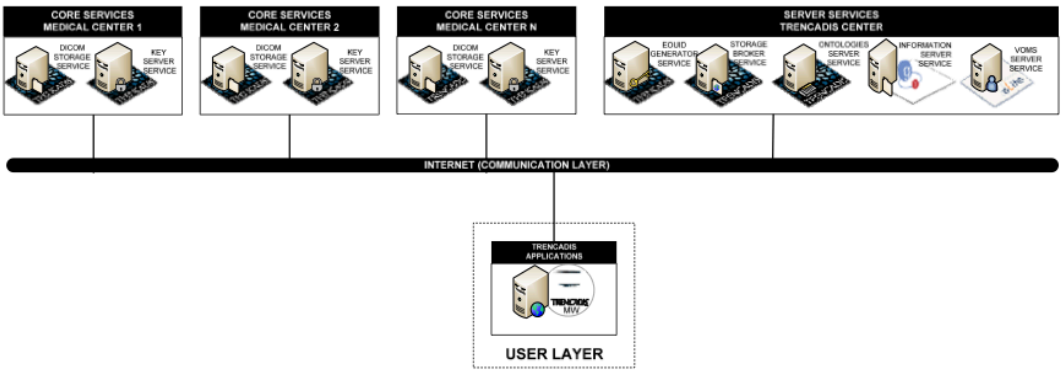

Figura 8 - Infraestructura de TRENCADIS.

2.2.1.1. CORE Services.

El objetivo de esta subcapa es la de posibilitar la comunicación con los recursos gLite de la infraestructura médica (dichos servicios son desplegados en la propia

infraestructura del hospital), permitiendo el almacenamiento de grandes cantidades de datos en entornos distribuidos. Soporta distintos tipos de *BackEnd*, proporcionando un interfaz uniforme a las capas superiores. Está formado por el servicio de almacenamiento DICOM (*DICOM Storage Service*) y el servicio del servidor de claves (*Key Server Service*). Antes de pasar a definirlos es necesario describir que existe un conjunto de herramientas base para el correcto despliegue de los servicios Grid, compuesto por:

- Globus 4 Toolkit.
- JDK 1.6
- Ant 1.8

DICOM Storage Service.

Se trata de un servicio que ofrece la interfaz para compartir objetos DICOM (tanto imágenes DICOM como informes estructurados DICOM-SR). Está compuesto por:

- ***DICOM Storage Grid Service***: Este servicio proporciona diferentes APIs para poder conectarse al *Indexer* y a los distintos *BackEnd*, dependiendo de cual sea su implementación. Se encuentra desplegado en el contenedor de servicios proporcionado por Globus 4.
- ***Indexer***: Este componente indexa los datos contenidos en los informes estructurados DICOM-SR. Actualmente se encuentra en producción con dos implementaciones: AMGA y PostgresSQL. Esta tesis de master desarrolla una nueva implementación basada en un modelo de grafos, e implementada sobre la base de datos noSQL denominada Neo4j, por lo que la mayoría del trabajo realizado se encuentra en dichos componentes.
- ***BackEnd***: Este componente almacena (de forma cifrada) las imágenes DICOM, así como los informes estructurados DICOM-SR asociados a dichas imágenes. Puede hacer uso de distintos sistemas para ello, como GridFTP, o LFC, etc. Actualmente se está trabajando en la posibilidad de almacenarlos en la nube, siguiendo el estándar propuesto por SNIA en CDMI (*Cloud Data Management Interface*).

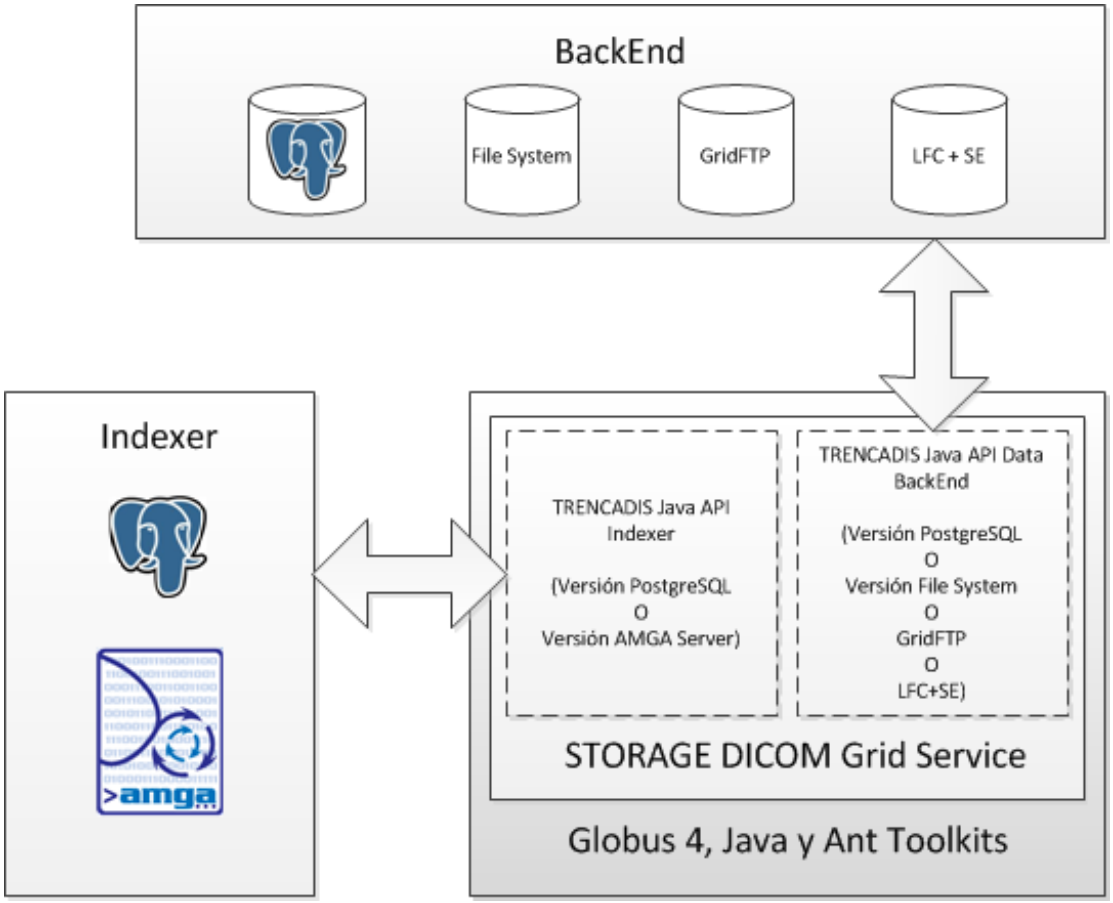

Figura 9 -DICOM Storage Service.

Key Server Service.

Se trata de un servicio que proporciona las claves para la encriptación de los objetos DICOM, por lo que es necesario para la integración con el VO al que pertenece la infraestructura médica. Está compuesto por:

- ***Key Server Grid Service***: Se trata de un servicio Grid desplegado en el contenedor de servicios proporcionado por Globus 4, el cual conecta con el componente *SQL Key Database*.
- ***BackEnd***: Este componente almacena las claves usadas para la encriptación de los distintos objetos DICOM. Estas claves se encuentran almacenadas en una base de datos relacional.
- ***SQL Keys Database***: Base de datos relacional instalada en el *Backend*.

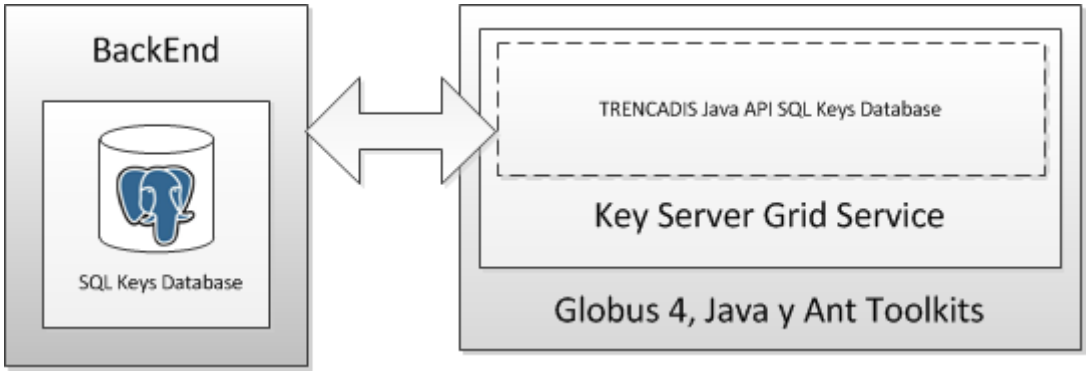

Figura 10 - Key Server Service.

2.2.1.2. SERVER Services.

Los servicios de servidor (*SERVER Services*) es un conjunto de servicios externos a los centros médicos, y desplegados en el centro TRENCADIS (actualmente la UPV). Los servicios son:

- *EQUID Generator Service.*
- *Storage Broker Service.*
- *Ontologies Server Services.*
- *Information Server Service.*
- *VOMS Service.*

EQUID Generator Service.

Servicio Grid desplegado en el contenedor de servicios proporcionado por Globus 4. Este implementa la lógica para la generación del identificador único del objeto encriptado (*Encrypted Object Unique Identifier*).

Storage Broker Service.

Servicio Grid desplegado en el contenedor de servicios proporcionado por Globus 4. Este implementa la lógica de la distribución de las consultas sobre los metadatos de los informes estructurados DICOM-SR distribuidos a lo largo de los distintos *DICOM Storage Services* pertenecientes al sistema, y recupera los resultados.

Ontologies Server Services.

Dicho servicio almacena y publica las ontologías disponibles en el sistema TRENCADIS en formato de plantillas, donde los metadatos se encuentran organizados.

Cada ontología contiene un árbol de contenidos DICOM-SR, donde sus relaciones semánticas se encuentran organizadas en una estructura de árbol. A través de este esquema, toda la estructura del informe y su contenido puede ser utilizado para la indexación y búsqueda de imágenes DICOM.

TRENCADIS utiliza XML para la especificación de las ontologías, ya que es lo suficientemente expresivo para la definición de términos y reglas a nivel de datos de DICOM-SR.

Las distintas plantillas contienen atributos, restricciones y relaciones entre campos, siguiendo la normativa propuesta por el estándar DICOM-SR.

Está compuesto por los siguientes componentes:

- ***Ontologies Server Grid Service***: Se trata de un servicio Grid desplegado en el contenedor de servicios proporcionado por Globus 4. Dicho servicio Grid usa una API para conectarse con la base de datos de ontologías (*SQL Ontologies Database*), instalado en el *BackEnd*.
- ***BackEnd***: Este componente almacena las ontologías usadas para organizar los datos DICOM.
- ***SQL Ontologies Database***: Este componente se trata de una base de datos relacional, instalada en el *BackEnd* (en este caso PostgresSQL).

Information Server Service.

Este servicio forma parte del sistema de monitorización y descubrimiento (MDS4) de Globus 4.

VOMS Service.

Este servicio forma parte del *middleware* gLite para la computación Grid.

2.2.2. Despliegue de TRENCADIS en la nube.

Actualmente existe una propuesta para poder desplegar TRENCADIS utilizando la virtualización y las técnicas de computación en la nube [9]. Dicha propuesta evita el despliegue innecesario de servicios, reduciendo la cantidad de esfuerzo que se requiere para instalar y mantener nuevos servicios TRENCADIS. Además de todo esto, proporciona mecanismos de control y gestión de requisitos de rendimiento y fiabilidad, donde la elasticidad de la nube juega un papel muy importante, asignando y desasignando recursos según los distintos picos de trabajo o requerimientos de las aplicaciones.

El sistema ha sido probado y desplegado con éxito emulando 3 centros médicos y un centro TRENCADIS, y utilizando como plataforma de nube Amazon EC2, y como plataforma de virtualización KVM, obteniendo unos buenos resultados.

Para el despliegue en la nube se han definido distintos componentes de la arquitectura de la plataforma. Dichos componentes son:

- Repositorio de imágenes de máquinas virtuales, denominado RVMI (*Repository of VMIs*).
- Repositorio de componentes software, denominado RSC (*Repository of Software Components*).
- Repositorio de descripciones de servicio, denominado RSD (*Repository of Service Descriptions*).
- Instanciador de infraestructura (*Infraestructure Instantiator*).
- Gestor de máquinas virtuales, denominado VMM (*Virtual Machine Manager*).

3. Objetivos.

Tras presentar la motivación de este trabajo y el estado de las tecnologías actuales relacionadas con el mismo, pasamos a presentar el objetivo de la tesis.

El objetivo principal de este trabajo es diseñar e implementar un nuevo modelo basado en grafos para indexar la información incluida en informes estructurados DICOM-SR, asociados a objetos DICOM (imágenes, videos, señales …), que permita desplegar infraestructuras Grid avanzadas TRENCADIS para escenarios enmarcados en el contexto de Big Data. Además, la inclusión de este nuevo modelo debe ser totalmente compatible con el modelo ya existente.

Los subobjetivos que subyacen del objetivo principal planteado son los siguientes:

1. Estudio general de las tecnologías noSQL que dan soporte a modelos basados en grafos para contextos de Big Data. Además de un estudio minucioso sobre la tecnología Neo4J, la cual será la que se utilizará finalmente para la implementación del modelo.
2. Diseñar un modelo basado en la teoría de grafos que modele la estructura arbórea de los informes estructurados DICOM-SR y permita almacenar y recuperar información de sus instancias. El diseño tomará en consideración aquellas características propias que ofrece la tecnología Neo4J, que permitan una mejora en las operaciones (creación de índices, cachés etc…).
3. Definir e implementar las operaciones básicas CRUD (*Create, Read, Update and Delete*) sobre el nuevo modelo, siempre cumpliendo con las propiedades ACID (*Atomicity, Consistency, Isolation and Durability*). Estas operaciones se proporcionarán a través de una nueva API en TRENCADIS, que se denominará *DSRNeo4jManager*, y permitirá sustituir el indexador actual basado en AMGA y denominado *DSRAMGAManager*.
4. Facilitar el desarrollo de futuras implementaciones en nuevos sistemas de almacenamiento a través de nuevas APIs.
5. El nuevo servicio ha de ser totalmente compatible con el servicio *Storage Broker*, manteniendo el reparto de consultas por los distintos repositorios (en este caso en Neo4j), y combinando los distintos resultados antes de presentarlos. Esto permitirá poder utilizar consultas con repositorios de distinto tipo.

4. Desarrollo del proyecto.

Para el desarrollo del modelo basado en grafos se ha aplicado un paradigma de desarrollo de software basado en el modelo Clásico en Cascada:

1. Estudio, análisis y diseño.
2. Desarrollo e implementación.
3. Pruebas.
4. Conclusiones.

Se han realizado dos iteraciones completas, dado que se han generado 2 modelos distintos, cada uno de ellos partiendo de ideas de funcionamiento distintas tras los resultados obtenidos en las pruebas y sus conclusiones de los trabajos previos existentes en TRENCADIS y los resultados de la primera iteración.

4.1. Modelo V.1.0 : Representación completa de las estructuras de los informes estructurados y de las instancias de los mismos en un grafo.

4.1.1. Estudio, análisis y diseño del modelo versión 1.0.

Antes de comenzar a diseñar el nuevo modelo basado en grafos, es necesario el análisis profundo de dos puntos. El primer punto es la representación en TRENCADIS de los informes estructurados DICOM-SR en documentos XML. El segundo punto es analizar los problemas asociados a los grafos a la hora de representar los informes DICOM-SR.

4.1.1.1. Representación de informes estructurados basados en DICOM-SR mediante el estándar XML.

DICOM es el estándar que la gran mayoría de centros médicos utilizan en las áreas de radiología y cardiología. El estándar no ha parado de crecer con nuevas especificaciones, que han permitido resolver las necesidades que han ido apareciendo a

lo largo de los años. Entre las nuevas especificaciones encontramos la incorporación de los informes estructurados DICOM-SR, los cuales han posibilitado la incorporación de información semántica a las imágenes médicas [10].

DICOM-SR se divide en dos partes bien diferenciadas: una cabecera de documento y un cuerpo del documento..

En el cuerpo del documento se encuentran los atributos para el elemento raíz del árbol que representa el informe estructurado, que incluye el código del título del informe y el propio árbol que representa el informe (*Content Tree*). En la figura 11 podemos ver su representación.

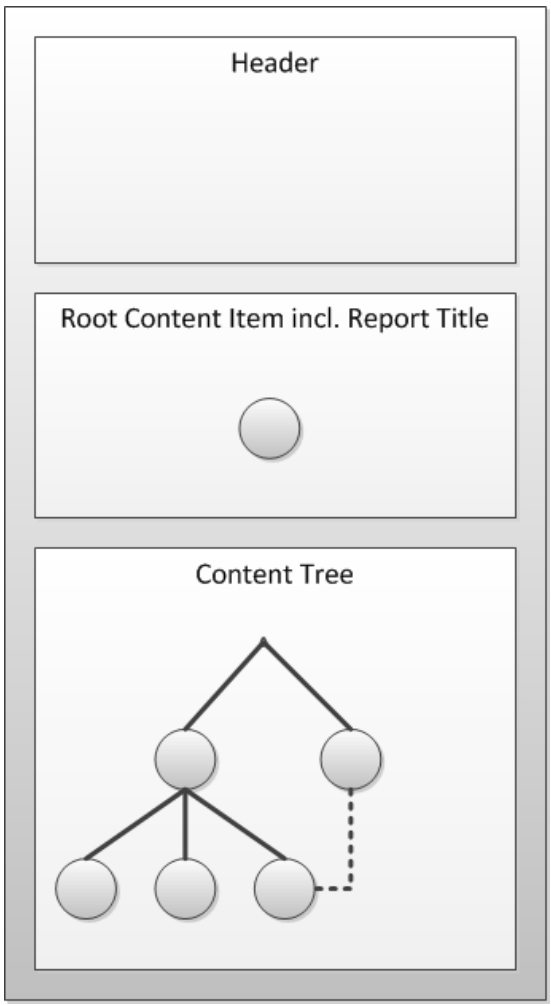

Figura 11 - Estructura del DICOM-SR.

El árbol que representa el informe tiene un elemento raíz, siendo este el elemento principal del mismo. Todos los elementos del árbol (*Content Item)* son de varios tipos, siendo el elemento raíz de tipo *CONTAINER*, encontrándose los demás elementos anidados a otros *Content Item*.

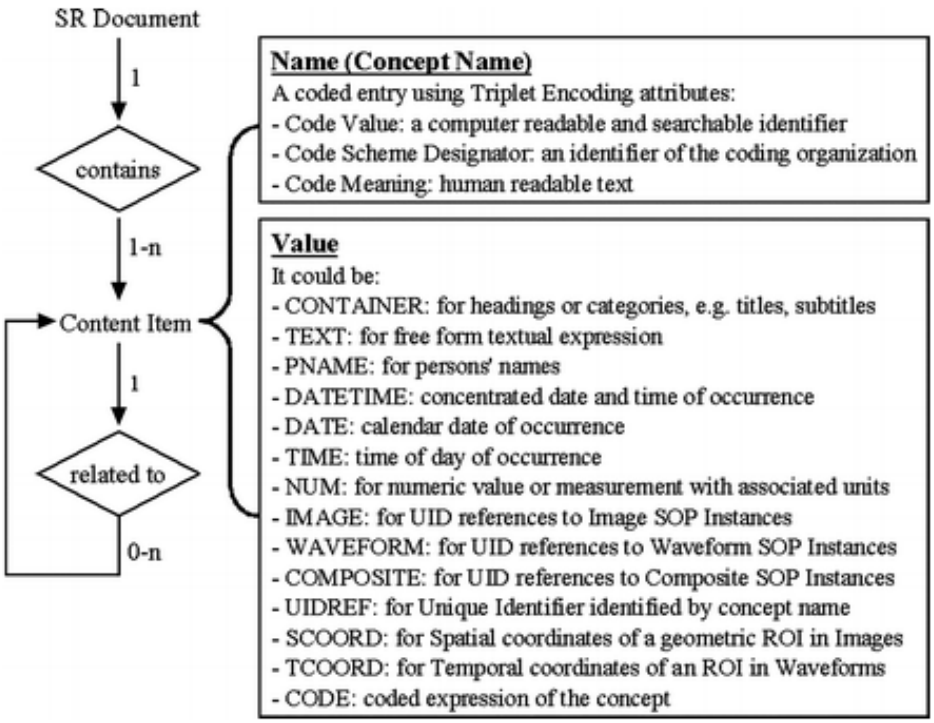

Figura 12 – Contenido de *Content Item* [34].

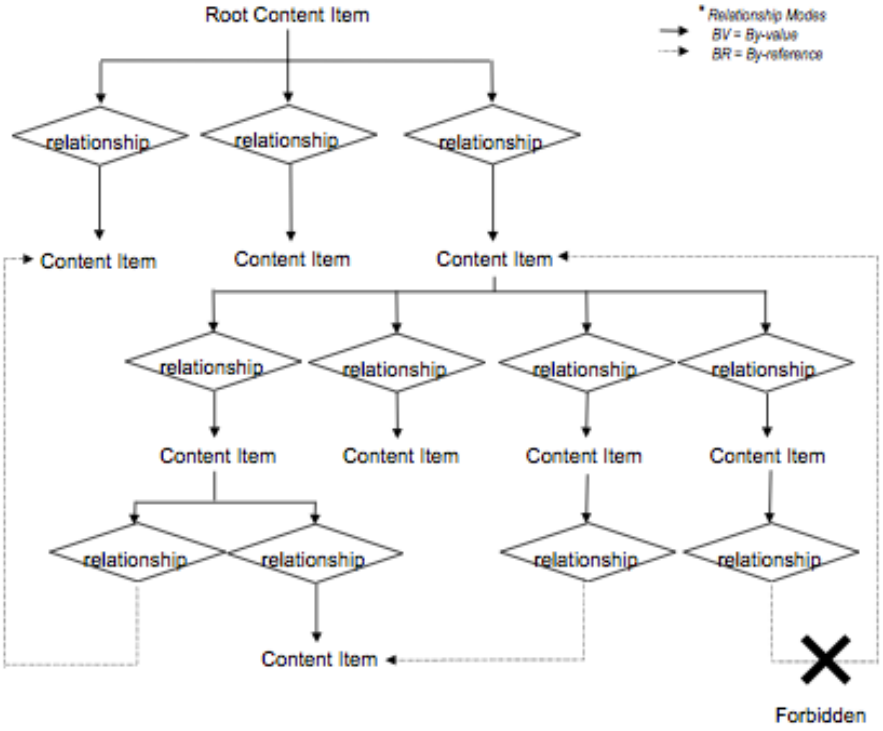

Figura 13 - Estructura del módulo de contenido [34].

Cada *Contem Item* contiene:

- Un par nombre/valor, que consiste en:
  - Un código de secuencia *Concept Name*, que es el nombre de un par nombre/valor o su cabecera.

  - Un valor (*TEXT*, *NUM*, *CODE*, etc.). En la figura 12 se puede observar los valores permitidos en el *Content Item*.
- Referencias a imágenes, formas de ondas u otros objetos compuestos, con o sin coordenadas.
- Relaciones con otros elementos:
  - Por valor, a través de secuencias de contenidos anidados.
  - Por referencia.

El *Concept Name* específica el concepto o significado del *Content Item* de forma codificada (por ejemplo una cadena de texto que describa un hallazgo asociado). Está compuesto por 3 cadenas de texto:

- *Code Value*: Valor escogido dentro de un diccionario para especificar el significado del *Content Item*.
- *Code Schema*: Identificador del diccionario de códigos utilizado.
- *Code Meaning*: Nombre o descripción del *Concept Name*.

Podemos observar un ejemplo de *Content Item* en el capítulo 3 de [10], donde el tipo *TEXT* se puede observar como en la figura 14.

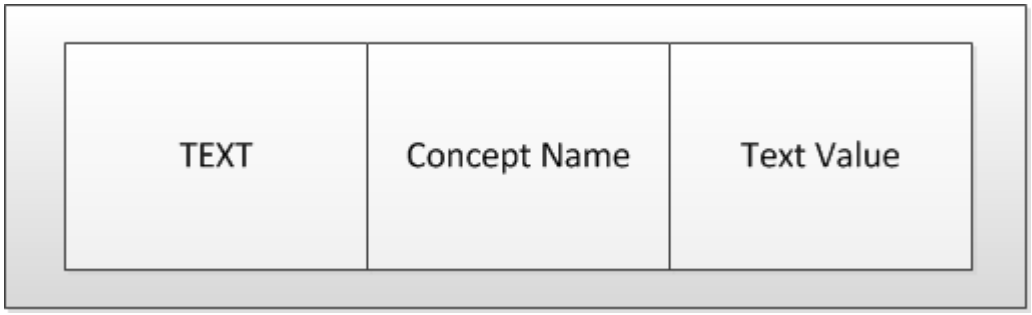

Figura 14 - Tipo TEXT.

Donde el valor del tipo TEXT puede ser escrito como:

```
<TEXT:(,,“Finding”)=”Large, irregular mass”>
```

O también se puede precisar de la siguiente forma:

```
<TEXT:(209001,99PMP,“Finding”)=”Large, irregular mass”>
```

Donde un volcado del objeto binario DICOM se podría visualizar de la siguiente manera:

```
(0040,a040) Value Type “TEXT”
(0040,a043) Concept Name Code Sequence
(fffe,e000) Item
(0008,0100) Code Value “209001”
(0008,0102) Coding Scheme Designator “99PMP”
(0008,0104) Code Meaning “Finding”
(fffe,e00d) Item Delimitation Item
(fffe,e0dd) Sequence Delimitation Item
(0040,a160) Text Value “Large, irregular mass”
```

Donde se puede ver que el *Code Value* es el valor **"209001"**, el *Code Schema* es el valor **"99PMP"** y el *Code Meaning* es el valor **"Finding"**.

A la hora de trabajar con el árbol no utilizaremos la representación binaria, sino una representación en el estándar XML propio de TRENCADIS, ya que simplifica mucho el tratamiento de la información. Podemos ver como se representa la información anterior en XML de la siguiente forma:

```
<TEXT>
  <CONCEPT_NAME>
     <CODE_VALUE>209001</CODE_VALUE>
     <CODE_SCHEMA>99PM</CODE_SCHEMA>
     <CODE_MEANING>Finding</CODE_MEANING>
  </CONCEPT_NAME>
  <VALUE>Large, irregular mass</VALUE>
</TEXT>
```

Existen otras características que deben cumplir los documentos basados en DICOM-SR, como que el informe estructurado se encuentre bien definido, de forma que un *Content Item* no debe contener elementos repetidos, y además estos han de posicionarse en orden correcto dentro del documento.

4.1.1.2. Consideraciones previas de los grafos.

Antes de abordar el diseño del modelo basado que grafos para la representación de informes estructurados definiremos algunos conceptos necesarios para la propuesta y construcción del mismo (y de sus instancias), ya que nos permitirán comprender las razones de ciertas decisiones de diseño a la hora de construir los distintos modelos.

En teoría de grafos, se llama camino a una secuencia de vértices dentro de un grafo tal que exista una arista entre cada vértice y el siguiente. Se dice que dos vértices están conectados si existe un camino que vaya de uno a otro, de lo contrario estarán desconectados. Dos vértices pueden estar conectados por varios caminos [33].

Consideramos un grafo no dirigido como un conjunto de nodos que se encuentran unidos por aristas sin dirección, como en la figura 15:

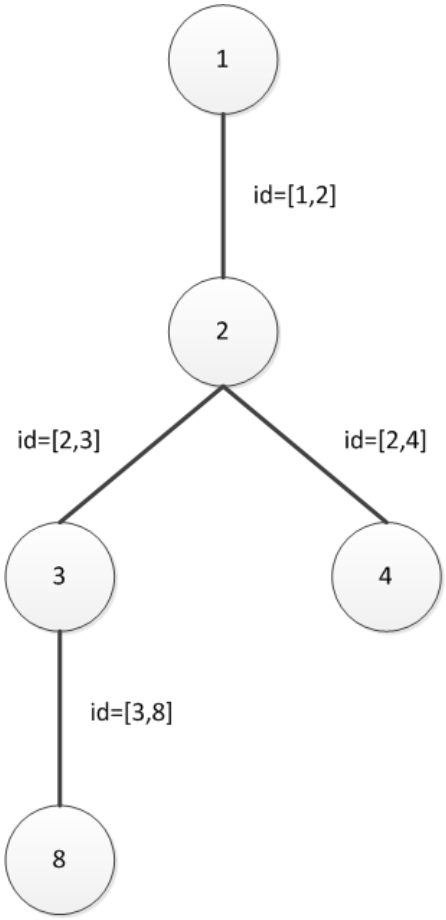

Figura 15 - Grafo inicial no dirigido.

En nuestro modelo, todos los nodos tendrán de identificador un número único, y será necesario que las aristas tengan, de la misma forma, un identificador propio, por lo que consideraremos como identificador de la arista el par de valores que une a los dos nodos, de forma que el identificador de la arista que une los nodos 1 y 2 será [1,2]. Esto nos permite saber cada arista que nodos une.

Esta necesidad de identificación de nodos y aristas surge por diferenciar los distintos elementos del informe estructurado dentro del fichero XML, así como los distintos niveles de jerarquía asociada a dichos elementos.

El problema con dicho método de identificación aparece cuando se produce una operación de fusión de grafos. Esta operación es la unión de 2 grafos, compartiendo de esta forma los nodos que son comunes entre ellos, y se producirá con las operaciones que crean nuevas estructuras en el grafo, como pueden ser dar de alta una nueva ontología, o una informe estructurado de un paciente en el sistema.

Supongamos el grafo de la figura 15 y el siguiente grafo contenido en la figura 16. En ambos grafos se comparte el nodo 2 y el nodo 3, pero pertenecen a distintos grafos. Si fusionamos ambos grafos el resultado de la operación aparece en la figura 17:

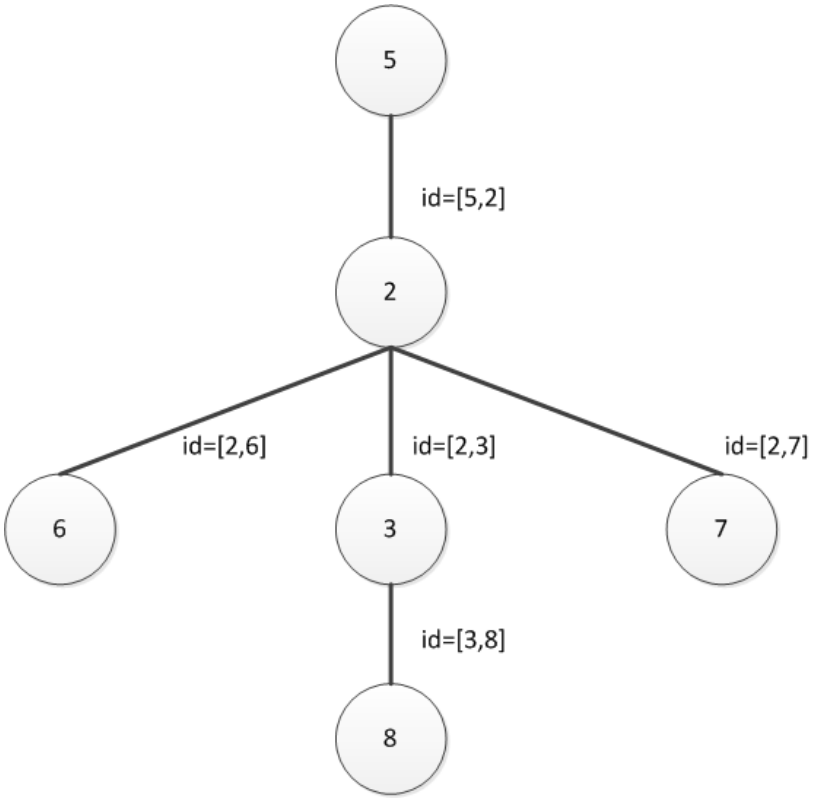

Figura 16 - Segundo grafo para realizar la fusión.

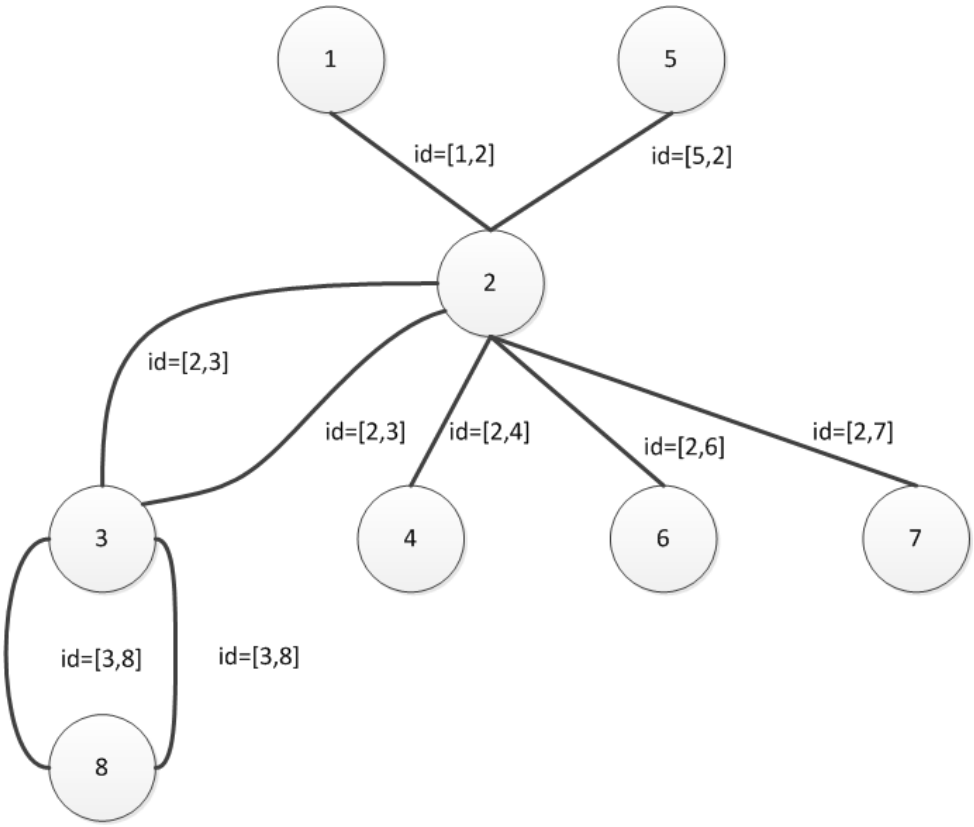

Figura 17 - Grafo resultado de la operación de fusión.

Como se puede observar en el nuevo grafo tras la fusión (figura 17) aparecen distintos problemas:

1. ¿Cómo el nodo 5 escoge el camino correcto hasta llegar al nodo 8?¿Podría ser el camino indicado en color rojo?

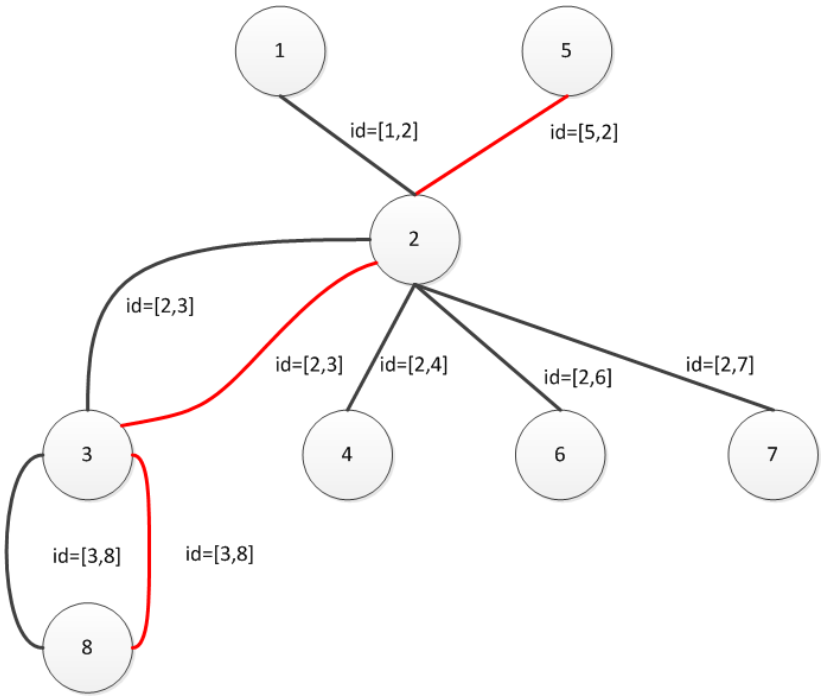

Figura 18 - Camino 1 entre el nodo 5 y el nodo 8.

¿O este otro?

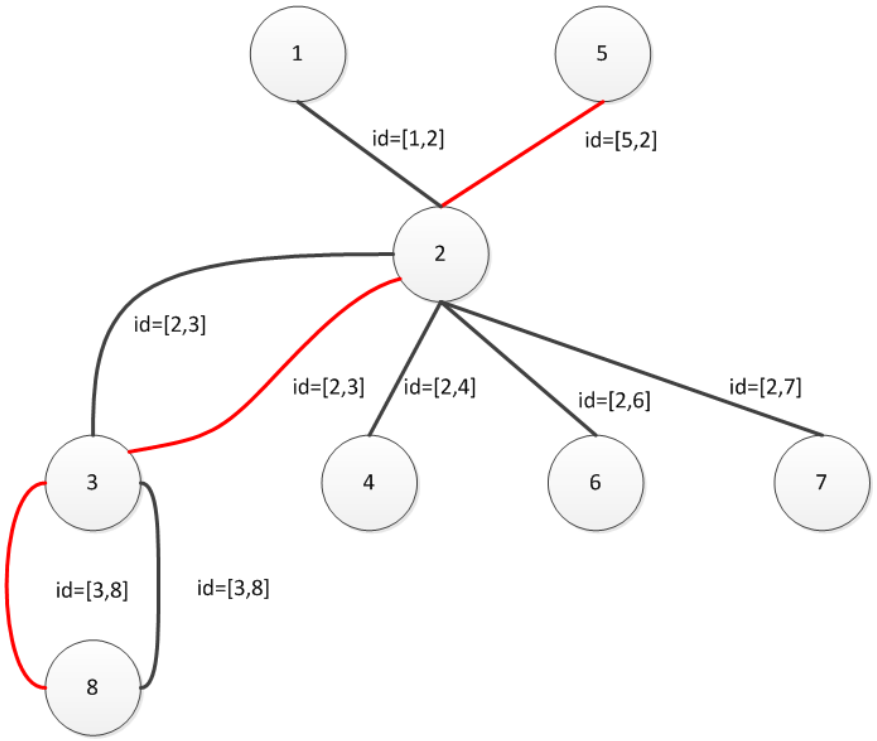

Figura 19 – Camino 2 entre el nodo 5 y el nodo 8.

2. ¿Qué diferencia hay entre las dos aristas con id [3,8] que aparecen entre los nodos 3 y 8?

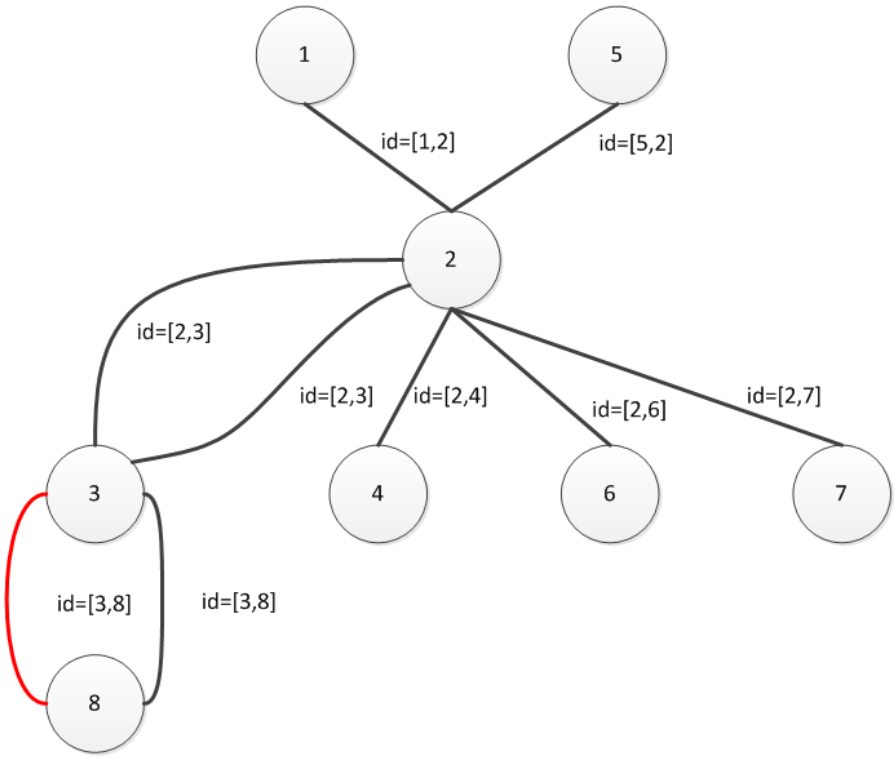

Figura 20 – Primera arista con id=[3,8].

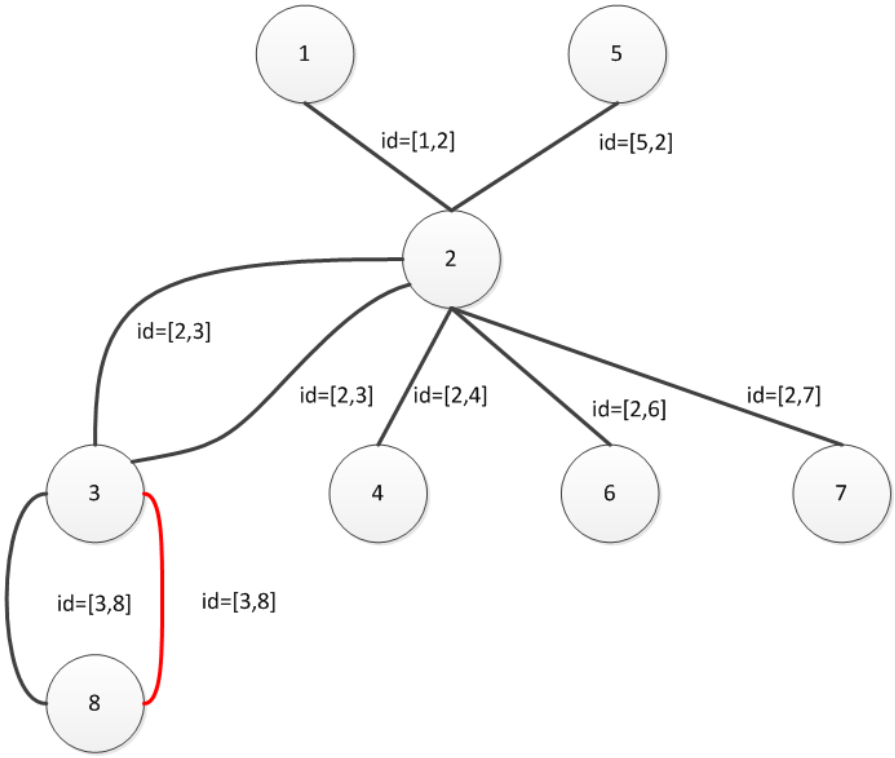

Figura 21 - Segunda arista con id=[3,8].

3. ¿Cómo podemos averiguar que camino el qué realmente contiene al nodo 7?

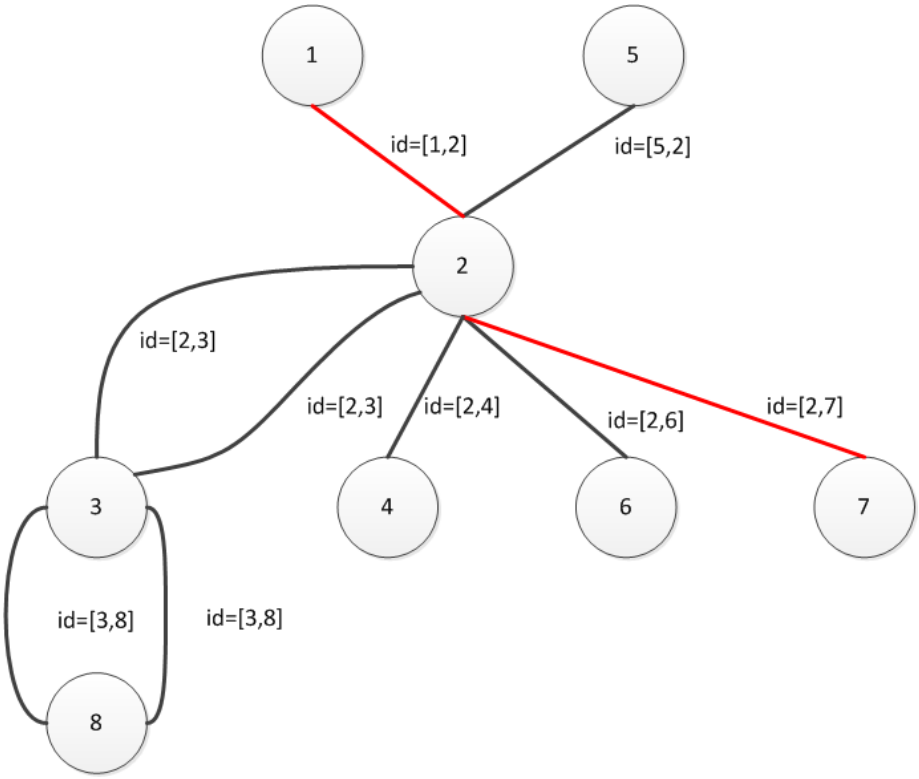

Figura 22 – Camino 1 que incluye al nodo 7.

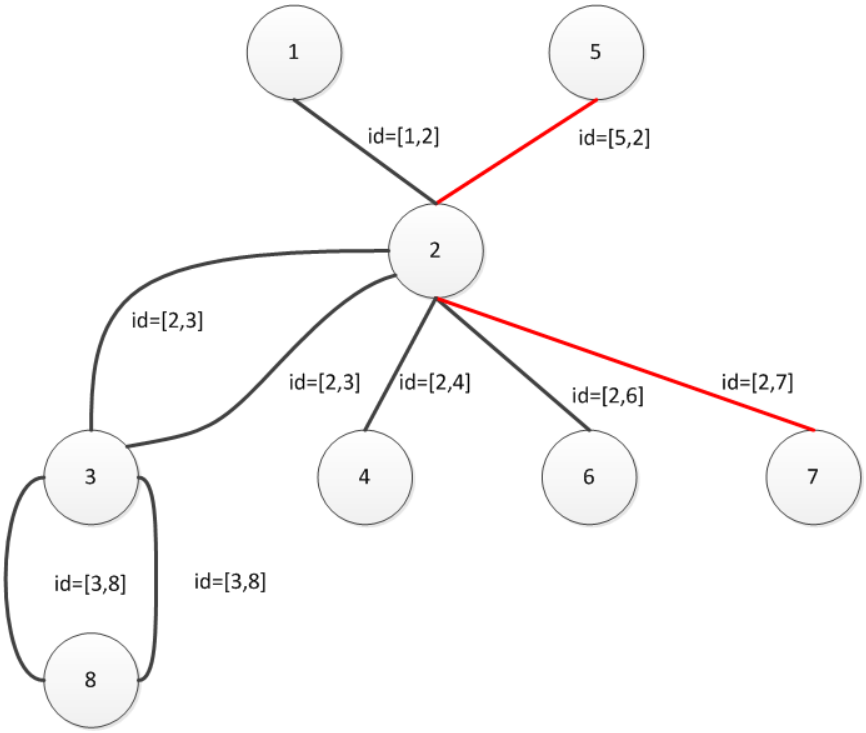

Figura 23 - Camino 2 que incluye al nodo 7.

Con esta situación planteada nos es imposible contestar a dichas cuestiones.

Para resolver este problema comenzaremos por cambiar el método de identificar las aristas. Ahora las identificaremos todas con un número único (al igual que los nodos), de forma que no existirán identificadores repetidos en las aristas (en la figura 24 podemos observar como queda el nuevo grafo):

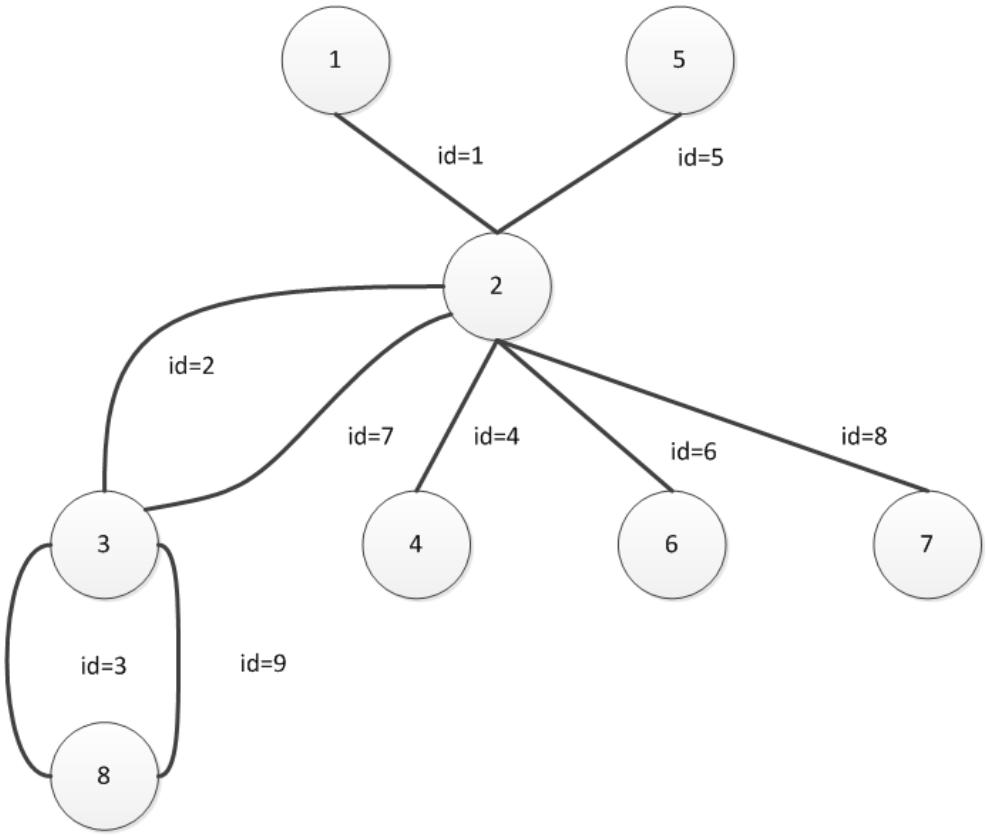

Figura 24 - Identificación única de las aristas.

Con esta solución ya tenemos las aristas claramente identificadas, y no es posible que exista una confusión entre ellas, aunque ya veremos más adelantes que esta propuesta con Neo4J no es viable por completo.

Tras tener las aristas identificadas de forma única crearemos un nuevo atributo denominado ***path***, el cual almacenará el identificador de la arista conforme va uniendo los nodos.

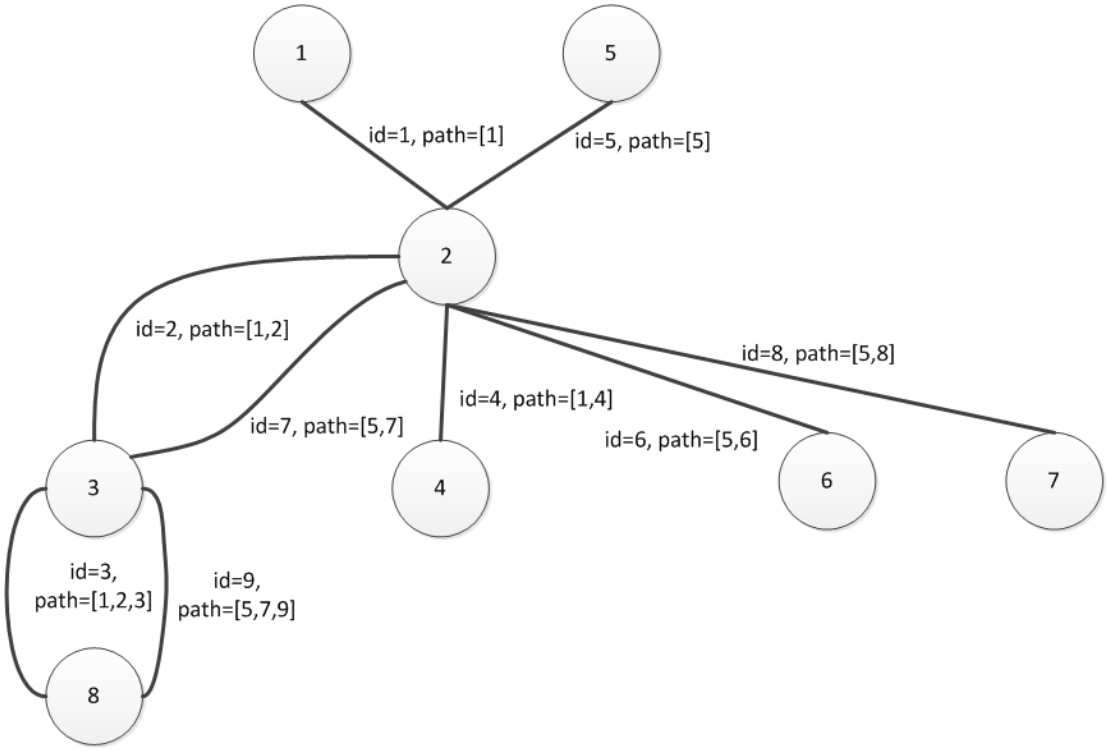

Figura 25 - Grafo con propiedad path conteniendo aristas.

Supongamos un camino existente entre los nodos 1 y 8:

- La arista que une los nodos 1 y 2 con identificador 1 almacenará en el atributo *path* el identificador de la arista 1.
- A continuación entre los nodos 2 y 3 la arista tendrá el identificador 2, y como dicha arista es parte del camino entre 1 y 8 almacenará el identificador de la arista 1 y la arista 2.
- Y para finalizar, entre los nodos 3 y 8 la arista recibirá el identificador 3 y el atributo *path* el identificador de las distintas aristas por las que pasa el camino (ya que es la última arista), por lo que tendrá los valores 1, 2 y 3 que identifican a las distintas aristas.

De esta forma podríamos resolver los problemas mencionados:

¿Cómo el nodo 5 escoge el camino correcto para llegar al nodo 8? La solución en este caso viene dada por los atributos *path* de las aristas, indicados en color rojo en la figura 26.

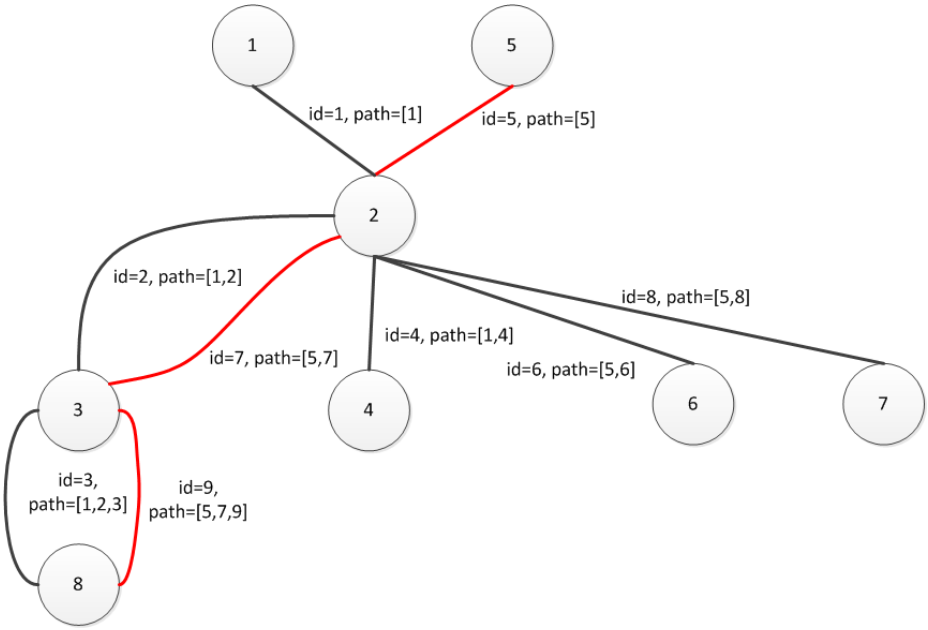

Figura 26 - Camino correcto desde el nodo 5 al nodo 8 a través de la propiedad *path*.

Pero almacenar los identificadores de las aristas genera un inconveniente, y es que el modelo va a trabajar principalmente con nodos, y no con aristas. Esto implica un problema en el diseño, ya que es necesario consultar en cada arista que nos situemos cuales son los nodos de los extremos de la misma, incrementando de esta forma el tiempo de respuesta de las consultas en el grafo. Por ello se decide optar por otra solución: en lugar de almacenar los identificadores de las aristas almacenamos los identificadores de los nodos que atraviesa el camino, como puede observarse en la figura 27.

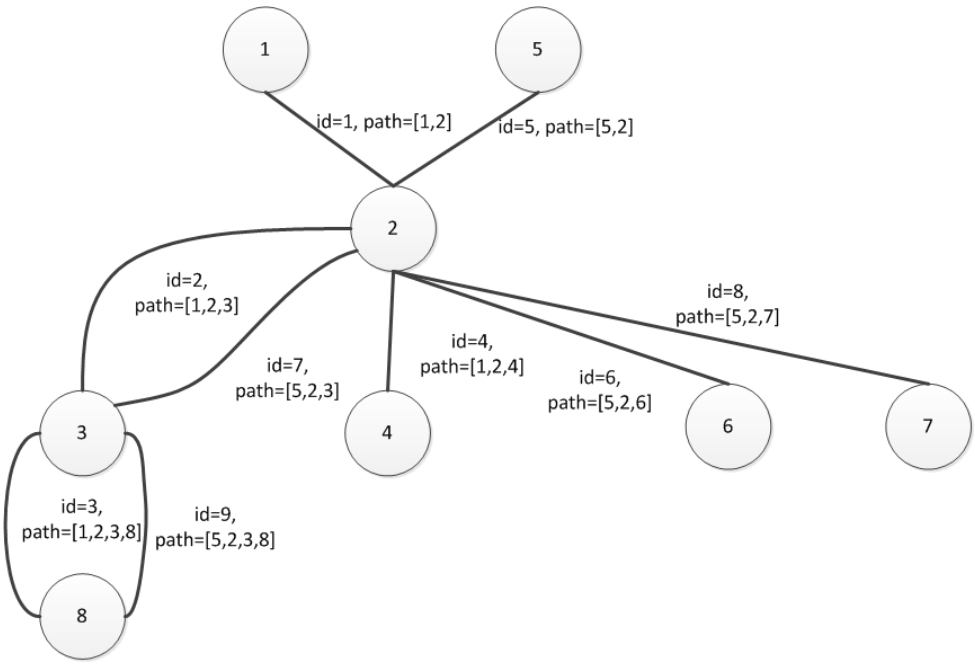

Figura 27 - Aristas con *path* almacenando los identificadores de los nodos.

Tras implementar dicho grafo en Neo4j podemos realizar una serie de pruebas para verificar si, efectivamente, la implementación realizada es una posible solución. Realizaremos, mediante *Cypher* (el lenguaje de consultas de Neo4j), una serie de pruebas para verificar su correcta implementación. El objetivo es buscar si existe un camino desde el nodo 5 hasta el nodo 8, a partir de la figura 27.

- Verificamos si, comenzando desde el nodo 5, existe alguna arista que contenga a dicho nodo (figura 28).

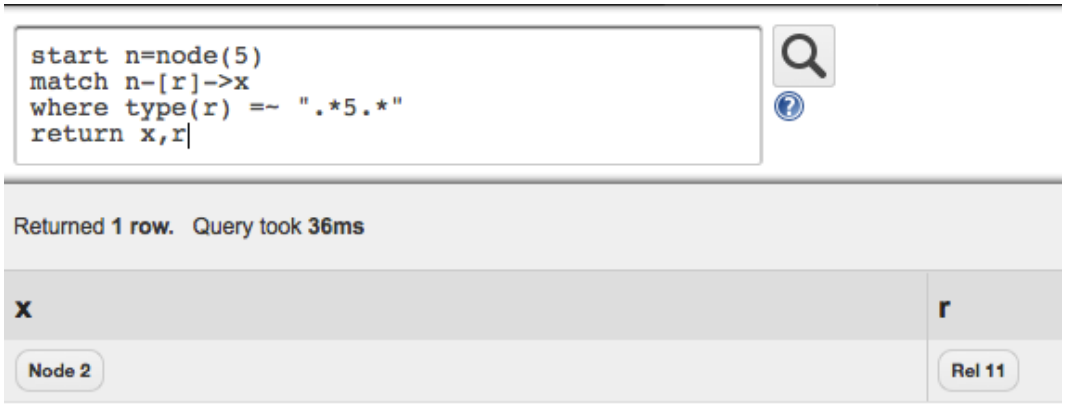

Figura 28 – Ejecución de consulta en *Cypher*.

Obtenemos como solución el nodo 2, o sea que existe una arista que tiene dicha información.

- Verificamos si, desde el nodo 2, existe alguna arista que contenga a dicho nodo (figura 29).

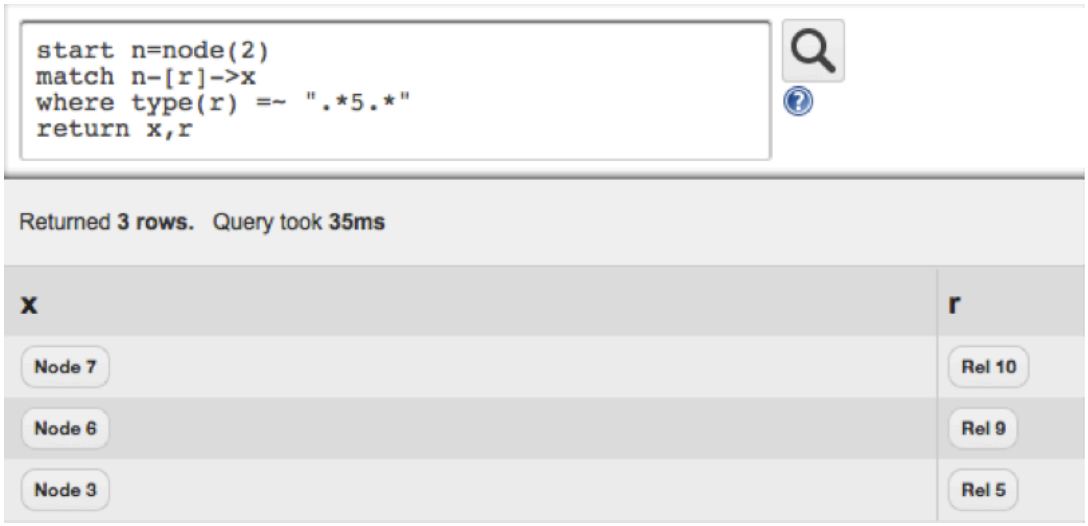

Figura 29 - Ejecución de consulta en *Cypher*.

Obtenemos como solución el nodo 7, 6 y 3, descartando el nodo 4, o sea que existen tres aristas que tienen dicha información.

- Verificamos si, desde el nodo 4, existe alguna arista que contenga a dicho nodo (figura 30).

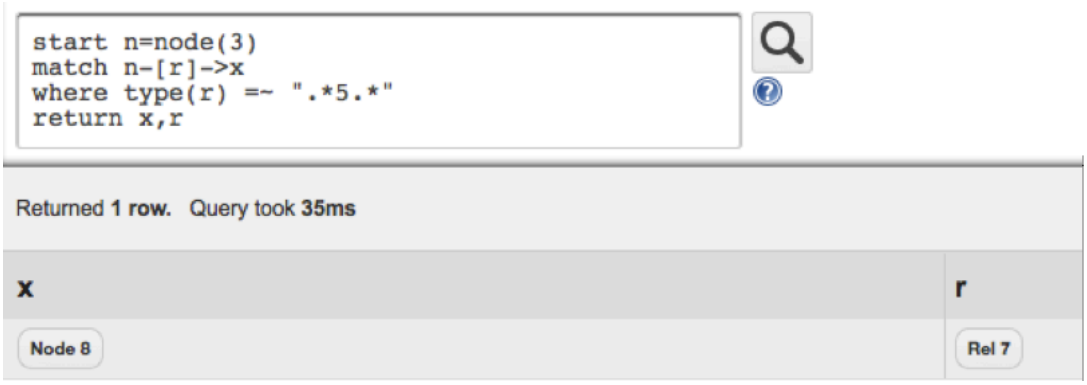

Figura 30 - Ejecución de consulta en *Cypher*.

Obtenemos como solución el nodo 8, confirmando de esta forma que existe un camino entre los nodos 5 y 8.

Otra cuestión que podemos resolver es la siguiente: ¿Tiene el nodo inicial 5 entre sus posibles caminos, en algún momento, al nodo 8? Esta cuestión puede ser resuelta obteniendo toda la lista de nodos, los cuales están relacionados con el nodo 5 (el atributo *path* de la arista correspondiente será quien los relacione, ya que dispone de la información). En la figura 31 podemos ver la comprobación de esto:

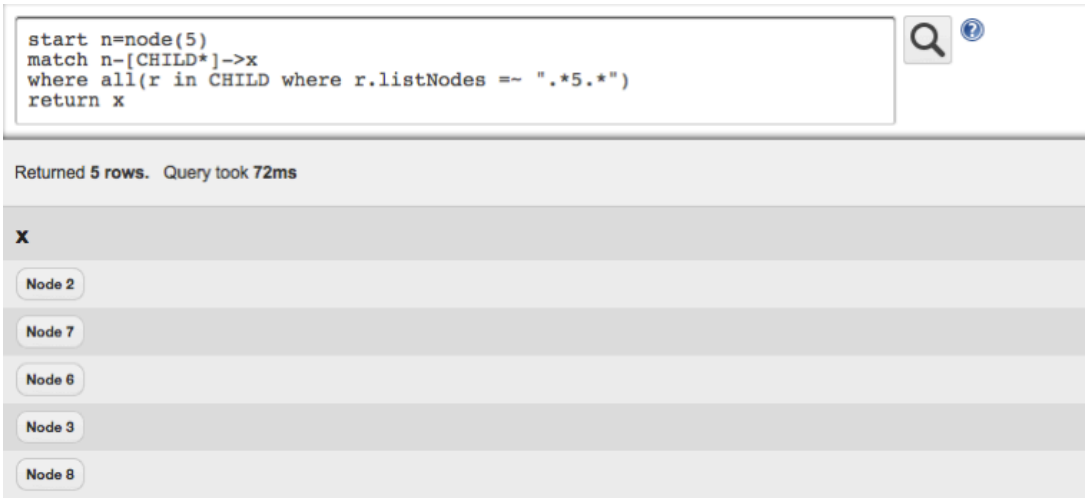

Figura 31 - Ejecución de consulta en *Cypher*.

Por lo que, sí podemos verificar que existe un camino por el que podemos llegar al nodo 8.

Aún así seguimos teniendo un problema en la solución propuesta. Imaginemos que tenemos el siguiente caso:

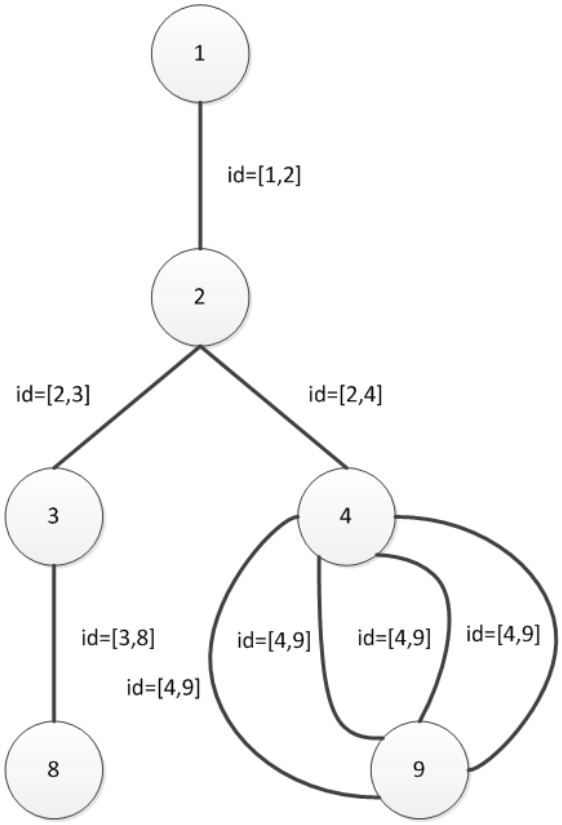

Figura 32 - Ejemplo de grafo con problema.

En dicho caso nos encontramos con que entre los nodos 4 y 9 existen 4 aristas con el mismo identificador. Al fusionar este nuevo grafo que aparece en la figura 32 con el grafo de la figura 16, utilizando la solución propuesta con anterioridad, tenemos lo siguiente:

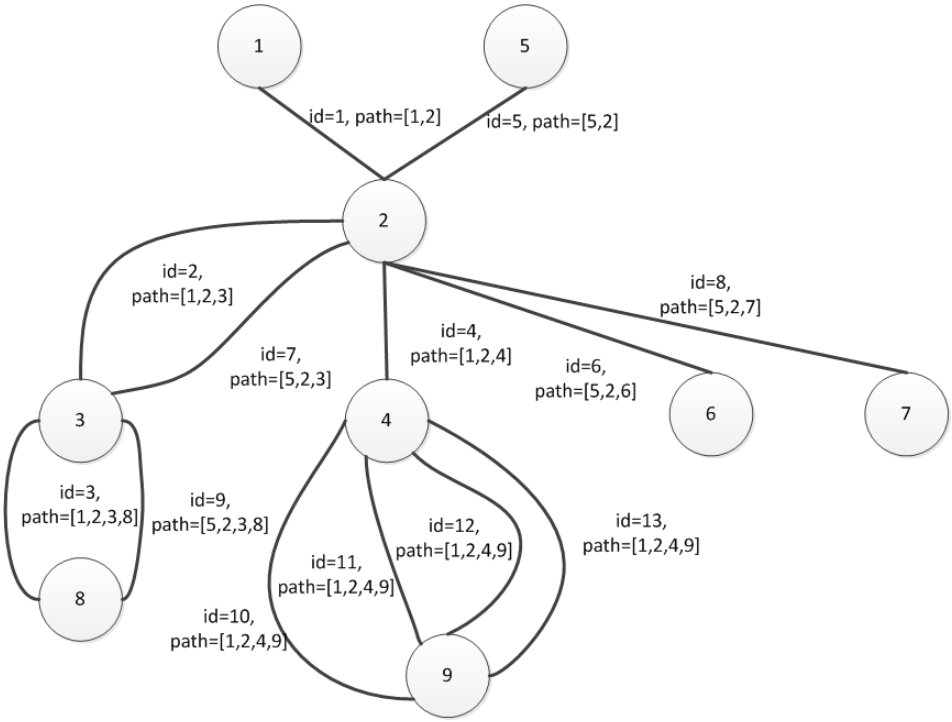

Figura 33 - Fusión de grafos de las figuras 32 y 16.

Se puede observar que las aristas 10,11,12 y 13 son las mismas, aunque originariamente pertenecen a distintos caminos dentro del grafo, por lo que la solución adoptada no es posible. Es necesario identificar de forma unívoca cada arista, por lo que introduciremos un nuevo componente a la hora de construir el atributo *path* denominado grado de multiplicidad.

El grado de multiplicidad será un número único, unido a cada identificador de nodo que se encuentra en el atributo *path*, junto con el carácter #, que separará el identificador de nodo del grado de multiplicidad.

Por lo que el nuevo grafo, aplicando el grado de multiplicidad, quedaría de la siguiente forma:

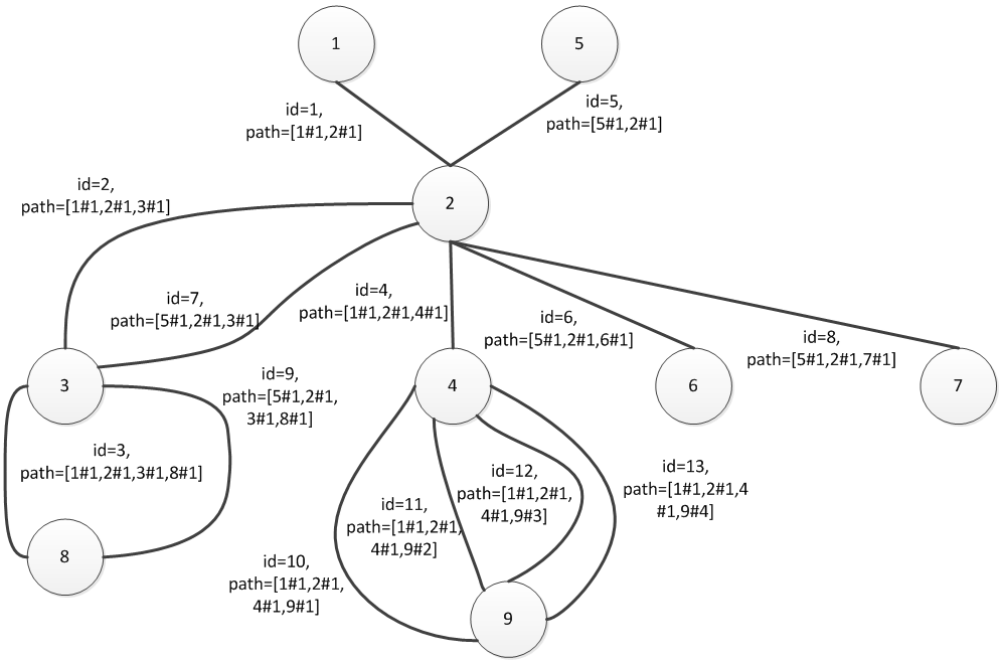

Figura 34 - Grafo final tras usar el grado de multiplicidad.

De forma que quedan perfectamente identificados los caminos en caso de multiplicidad n.

4.1.1.3. Análisis y diseño del modelo 1.0.

El modelo surge a partir de la descripción de la estructura de informes estructurados, la cual viene definida por plantillas implementadas en documentos XML, así como también de las instancias de los informes estructurados en base a estructuras definidas. Por ello, todo en su conjunto (estructura + datos) serán representados en forma de nodos y relaciones (aristas).

Con ello se conseguirá un modelo en forma de grafo, con una estructura muy concreta, aunque todos los datos se encuentran totalmente desestructurados, de forma que uno por sí mismo (un nodo o relación) no tiene sentido, manteniendo así toda la

información de los ficheros XML con el fin de poder reconstruir un fichero en cualquier momento realizando las consultas adecuadas.

Solo se podrá recorrer el grafo de una forma muy concreta (se trata de un grafo dirigido), posibilitando obtener la información de forma correcta y con sentido. Al final se tendrá un gran grafo, donde tendremos un subgrafo que mantendrá la estructura del informe y n subgrafos que mantendrán las instancias de estos en base a una estructura.

Los informes estructurados se encuentran en formato XML, de forma que la información está perfectamente estructurada, en partes bien definidas mediante etiquetas, formando de esta forma los elementos, y proporcionando cada etiqueta un sentido al concepto que representa. Por lo que, cada etiqueta, será un nodo de información en el grafo, que en este caso viene representado en forma de óvalo. Estos dispondrán de dos propiedades:

- ***typeNode***: tipo de nodo que representa.
- ***idOwn***: identificador interno único del nodo.

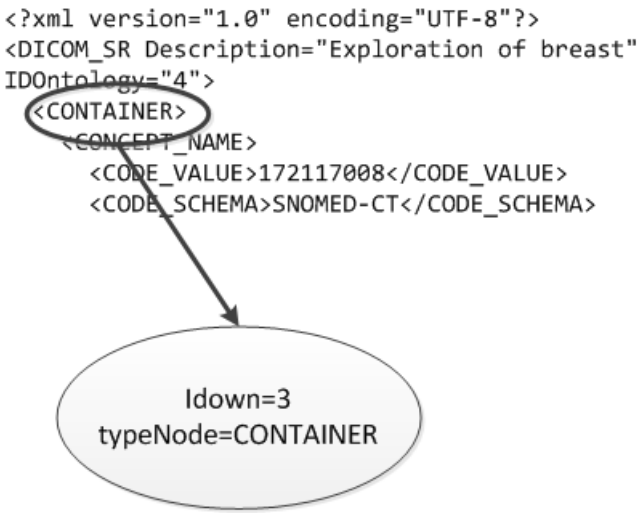

Figura 35 - Identificación de un nodo.

Existen distintos tipos de nodos o vértices según el uso que vaya a tener:

- **Nodo de estructura**: Dicho nodo va a mantener una entrada perteneciente a la estructura de un informe estructurado. Dentro de los nodos de estructuras distinguiremos 3 tipos de nodos:
    - **Nodo de estructura simple**: Contendrá un nodo de una entrada simple de la estructura. Un ejemplo sería la entrada *CONTAINER, DATE, CONCEPT_NAME*, etc.
    - **Nodo de estructura agrupada**: Contendrá un nodo ficticio que será el nodo padre de una agrupación de nodos. Un ejemplo sería el nodo ficticio *AGR_CARDINALITY*, que agrupará los distintos nodos *CARDINALITY* que se van a crear.
    - **Nodo de estructura compleja**: Contendrá un nodo de una entrada que posee atributos en la misma. Dicho nodo será agrupado en su nodo de estructura agregada correspondiente. Un ejemplo sería el nodo

CARDINALITY, el cuál dispone de los atributos *max* y *min* junto al mismo.

- **Nodo valor**: Dicho nodo va a mantener una entrada perteneciente a un informe estructurado en concreto de un paciente. Un ejemplo sería el nodo con valor **"28/03/2013"**, que representa una fecha.
- **Nodo mixto**: Se trata de un nodo que en su origen, o bien era de tipo nodo valor, o bien era un nodo de estructura simple, solo que en algún momento ha tenido un uso distinto debido a la reutilización de los nodos de la estructura.

Por lo que, estos niveles de jerarquía, han de estar representados en la solución propuesta, y para ello se hace uso de la otra característica propia del modelo de grafos a utilizar: las relaciones.

Las relaciones representarán la jerarquía de la información, o lo que es lo mismo, representarán la relación existente entre una etiqueta que se encuentra contenida dentro del ámbito de otra.

```
<CONTAINER>
      <CONCEPT_NAME>
            <CODE_VALUE>172117008</CODE_VALUE>
            <CODE_SCHEMA>SNOMED-CT</CODE_SCHEMA>
            <CODE_MEANING>Exploración  de la Mama</CODE_MEANING>
      </CONCEPT_NAME>
```

Figura 36 - Representación XML de las jerarquías existentes.

De forma que el grafo obtenido tendrá una primera aproximación como la que se describe en la siguiente figura (ya que dicho representación va a ser refinada con posterioridad):

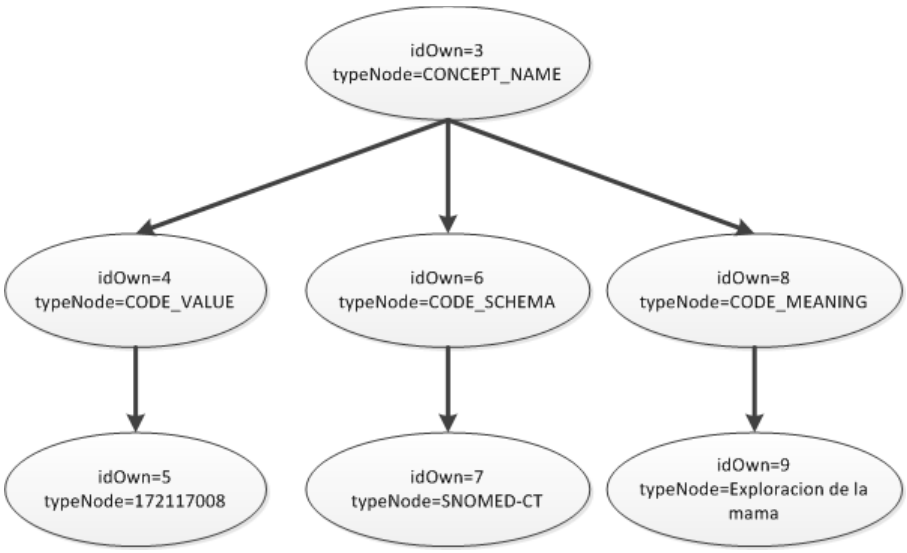

Figura 37 - Representación gráfica de las jerarquías existentes.

En este momento se ha obtenido un grafo dirigido, de forma que los nodos tendrán un origen y un destino. El problema surge cuando una nueva estructura es dada de alta, como puede ser la de la figura 38:

```
<CONCEPT_NAME>
      <CODE_VALUE>118522005</CODE_VALUE>
      <CODE_SCHEMA>SNOMED-CT</CODE_SCHEMA>
      <CODE_MEANING>Identificador</CODE_MEANING>
</CONCEPT_NAME>
```

Figura 38 - Alta de nueva estructura.

El modelo seleccionado permite reutilizar los nodos existentes para crear nuevas relaciones, por lo que se producirá la siguiente situación tras implementar la nueva jerarquía de nodos (aparecerá de gris las nuevas relaciones) en la figura 39:

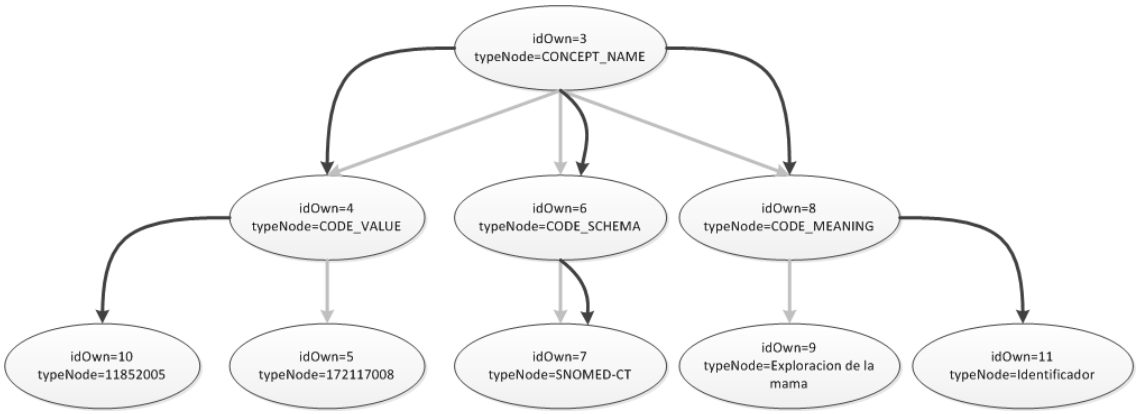

Figura 39 - Representación gráfica de ambas estructuras reutilizando nodos.

Tal y como describe el estándar DICOM-SR se usan los elementos *CODE_VALUE* y *CODE_SCHEMA* para implementar los elementos que pertenecen al mismo nivel de jerarquía, y para ello la unión de ambos creará un nuevo nodo identificador que realizará dicho cometido.

Para ello, en el grafo, a partir del nodo valor del nodo estructura *CODE_VALUE* se jerarquizará el nodo valor del nodo estructura *CODE_SCHEMA*, creando así el nodo identificador, siendo el mismo la unión del valor de ambos nodos valor. A partir de dicho nodo identificador se jerarquizarán todos los demás nodos o elementos. Esto genera una jerarquía de 3 niveles dentro del grafo.

Veamos un ejemplo en el caso de un informe para una exploración de mama, donde su *CODE_VALUE* tiene el valor *"172117008"* y su *CODE_SCHEMA* tiene el valor *"SNOMED-CT"*:

- El primer nivel de jerarquía hace referencia a la creación del nodo "*172117008*", que representa al nodo valor del *CODE_VALUE* de la exploración mamaria. Una vez creado el nodo valor, se creará una nueva

arista con el nodo estructura que mantiene la jerarquía con este valor (en este caso es del tipo *CONTAINER*), pasando así al segundo nivel.

- El segundo nivel de jerarquía hace referencia a la creación del nodo "*SNOMED-CT*", que representa al *CODE_SCHEMA* de la exploración mamaria. Tras crear el nodo valor se creará un nuevo nodo identificador, siendo este la unión de "*172117008*" y "*SNOMED-CT*", con el carácter "*_#_*" en medio como carácter de separación. Este nodo estará unido únicamente al nodo valor "*SNOMED-CT*". Mediante una nueva arista dirigida de nuevo al nodo *CONTAINER*, desde el nuevo nodo identificador "*172117008_#_SNOMED-CT*", creando así el tercer nivel de jerarquía.
- El tercer nivel de jerarquía hace referencia a qué nodos pertenecen al ámbito del nodo identificador con valor "*172117008_#_SNOMED-CT*". Y esta pertenencia se realiza en el atributo *path*.

Como se ha comentado en el punto anterior, las aristas contendrán los nodos que participan en los distintos caminos, indicando de esta forma el orden de la jerarquía. Dicha lista de nodos es almacenada en el atributo propio de la arista denominado *path* (comentado ya con anterioridad).

Por lo que la propiedad de la arista recibirá, como valor, el recorrido (o camino) de los nodos que va recorriendo (o atravesando) hasta llegar al nodo final, incluido este. Por lo que una relación entre dos nodos será caracterizada por dicha propiedad, indicando el conjunto de nodos desde donde se inicia la jerarquía de información, hasta llegar al nodo destino.

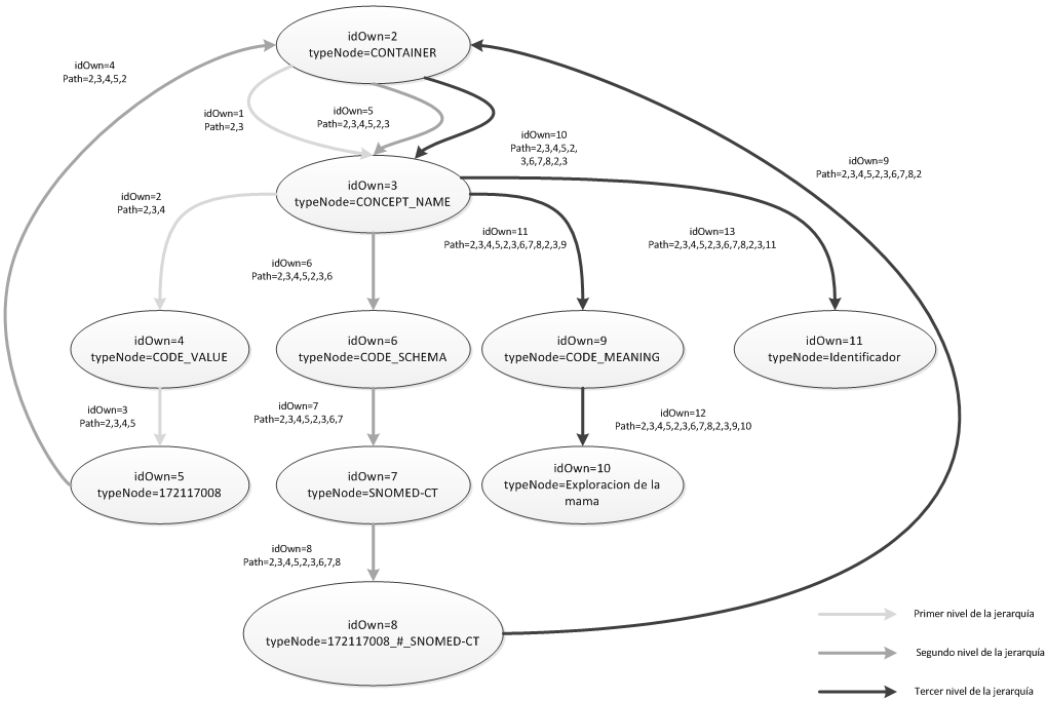

Figura 40 - Representación final del árbol de jerarquías.

De esta forma, al ser un grafo dirigido y etiquetado, con este sistema, podemos averiguar la "dirección" hacia donde se dirige la información, por lo que queda descrito la jerarquía semántica producida en el fichero XML.

Como ya se ha indicado en el punto 4.1.1.2, el problema del grado de multiplicidad sigue existiendo en este modelo propuesto. Aunque ya es posible identificar los distintos elementos pertenecientes jerárquicamente a otro elemento mediante el valor *path* de las aristas, si un elemento contiene dentro del mismo n elementos (o etiquetas) del mismo tipo no sabría distinguir entre ellas. Un ejemplo claro sería la figura 41 (es necesario indicar que cuando hablamos de elementos nos referimos a etiquetas XML que dan a lugar a nodos):

```
<CHILDS>
	<CONTAINER>
		<CONCEPT_NAME>...</CONCEPT_NAME>
		<PROPERTIES>...</PROPERTIES>
		<CHILDS>...</CHILDS>
	</CONTAINER>
	<CONTAINER>
		<CONCEPT_NAME>...</CONCEPT_NAME>
		<PROPERTIES>...</PROPERTIES>
		<CHILDS>...</CHILDS>
	</CONTAINER>
	<CONTAINER>
		<CONCEPT_NAME>...</CONCEPT_NAME>
		<PROPERTIES>...</PROPERTIES>
		<CHILDS>...</CHILDS>
	</CONTAINER>
</CHILDS>
```

Figura 41 - El problema de los múltiples elementos repetidos.

En este fragmento de informe estructurado, dentro del elemento "*CHILDS*" encontramos 3 elementos de tipo "*CONTAINER*". Cuando se crea la relación entre "*CHILDS*" y los 3 elementos "*CONTAINER*", al asignar a la propiedad "*path*", el valor que recibe en cada relación es idéntico, ya que el conjunto de nodos que forma la jerarquía, el nodo origen y el nodo destino, en los 3 casos, es el mismo (figura 42).

Para solucionar dicha situación es necesario identificar, de forma única, la relación entre el nodo origen y el nodo destino, por lo que se acude al grado de multiplicidad de los elementos (propuesto en el punto 4.1.1.2), de forma que se va a permitir identificar cada relación de forma unívoca.

El grado de multiplicidad se indica en cada nodo origen y nodo destino, de forma que el grado de los nodos destinos viene indicado por el grado de apariciones del elemento destino dentro del elemento origen (en este ejemplo tendremos que la primera aparición del elemento *CONTAINER* lleva el grado 1, la segunda aparición el grado 2, la tercera el grado 3, y así sucesivamente). El nodo origen recibe su multiplicidad. Para separar el identificador del nodo del grado de multiplicidad se utilizará el carácter #, por lo que una representación gráfica de la solución sería la de la figura 43.

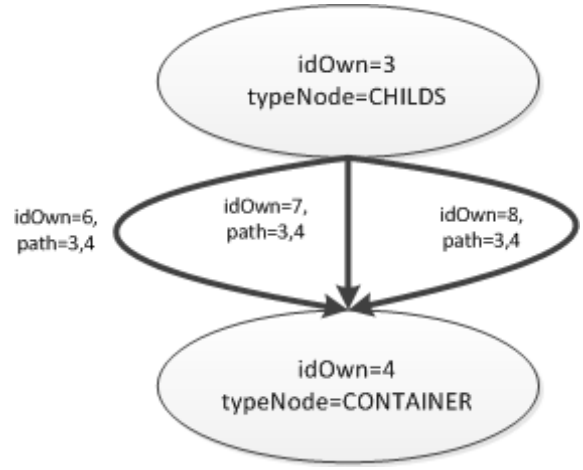

Figura 42 - Reutilización de nodos sin identificador de multiplicidad.

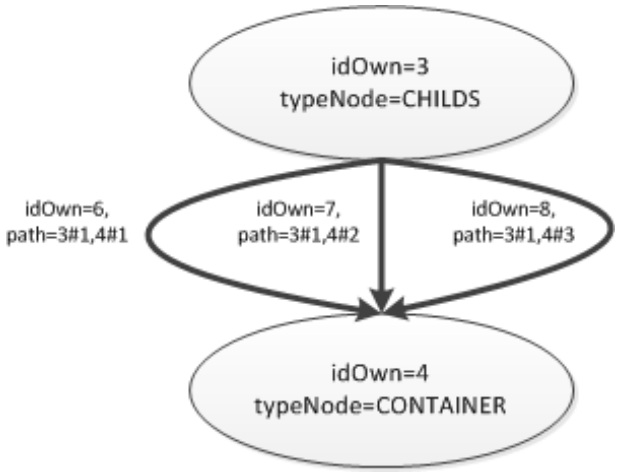

Figura 43 - Reutilización de nodos con identificador de multiplicidad.

En las figuras anteriores se puede observar la diferencia entre disponer del grado de multiplicidad y no disponer del mismo. Hay que tener en cuenta que los valores de la propiedad *path* son "arrastrados" a las siguientes relaciones, por lo que los grados de multiplicidad son "arrastrados" con ellos, permitiendo así identificar perfectamente a donde pertenece cada elemento.

Por lo que, las aristas tienen asociado, de forma muy definida, el atributo *path*, el cual indica 3 características (suponiendo el nodo n1 como nodo origen y el nodo n2 como nodo destino):

- El camino que ha seguido hasta llegar a n1.
- El identificador de n2.
- El grado de multiplicidad que le pertenece respecto a las otras aristas existentes que llegan al nodo n2.

Con lo que, aplicando dicho criterio a la figura 40 nos encontramos que el grafo será:

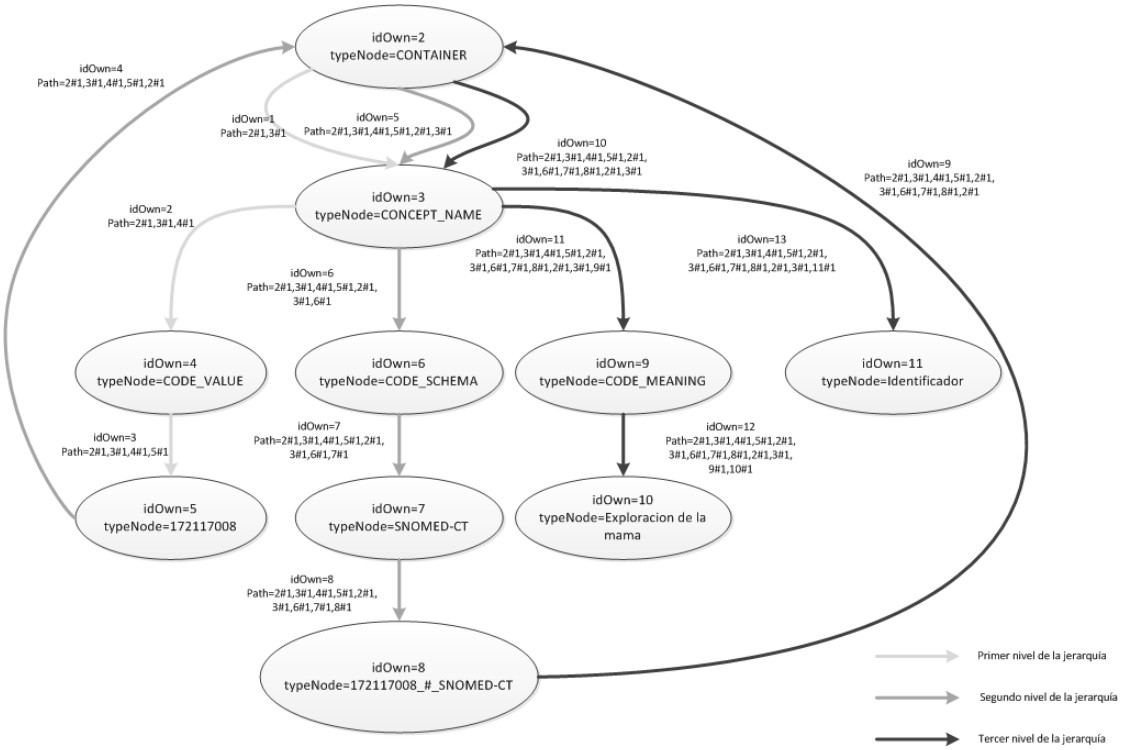

Figura 44 - Grafo con los 3 niveles de jerarquía aplicando el grado de multiplicidad.

Con dicho modelo presentado hasta este momento es posible almacenar las estructuras de los distintos informes estructurados disponibles, pero no las instancias de los mismos.

El proceso para insertar las instancias de los informes difiere a como se crean las estructuras por distintas razones:

- Es interesante crear una serie de relaciones directas entre el identificador del informe, el identificador del paciente, la fecha de la prueba y el identificador de la prueba (en este orden), debido a que proporciona una serie de respuestas a consultas muy básicas de una forma rápida y eficaz.
- Existen elementos que pertenecen a la estructura que se encuentran en los informes de datos, que no será necesario volver a darlas de alta (aunque a la hora de su recuperación sí que será necesario recuperarlas). En este caso los nodos son reutilizados creando nuevas relaciones a los mismos.
- Es necesario tener un nodo que soporte los valores propios de cada característica del informe (se irán creando nodos con valores según vayan siendo necesarios).

Siguiendo estas pautas, será necesario crear (con nodos y relaciones) una cabecera del informe. Se relacionará con los distintos elementos que mantiene el informe estructurado (que se encuentran en el fichero XML de la instancia) ya dado de alta, y relacionándolo con los valores asociados (ya sea porque existen o porque se crean) a la instancia del informe del paciente. Esto permite que, a partir del identificador del reporte, identificador del paciente, fecha de la prueba e identificador de la prueba (el conjunto de todo es la cabecera) es posible acceder a cualquier dato almacenado en el grafo recorriendo el camino correspondiente. Tras disponer de la cabecera del informe

almacenada correctamente, los datos serán almacenados siguiendo el patrón utilizado para la creación de estructuras a partir del informe estructurado.

Pongamos un ejemplo de una posible cabecera. Para ello utilizamos parte de una instancia de un informe estructurado de una exploración mamaria.

```
<?xml version="1.0" encoding="UTF-8"?>
<DICOM_SR IDOntology="4" IDReport="EQ_1_1" >
<CONTAINER>
  <CONCEPT_NAME>
    <CODE_VALUE>172117008</CODE_VALUE>
    <CODE_SCHEMA>SNOMED-CT</CODE_SCHEMA>
    <CODE_MEANING>Exploración  de la Mama</CODE_MEANING>
  </CONCEPT_NAME>
  <CHILDS>
    <DATE>
      <CONCEPT_NAME>
        <CODE_VALUE>399651003</CODE_VALUE>
        <CODE_SCHEMA>SNOMED-CT</CODE_SCHEMA>
        <CODE_MEANING>Fecha Informe</CODE_MEANING>
      </CONCEPT_NAME>
      <VALUE>28/03/2013</VALUE>
    </DATE>
    <TEXT>
      <CONCEPT_NAME>
        <CODE_VALUE>118522005</CODE_VALUE>
        <CODE_SCHEMA>SNOMED-CT</CODE_SCHEMA>
        <CODE_MEANING>Identificador</CODE_MEANING>
      </CONCEPT_NAME>
      <VALUE>EQ_1_1</VALUE>
    </TEXT>
    <TEXT>
      <CONCEPT_NAME>
        <CODE_VALUE>RID13159</CODE_VALUE>
        <CODE_SCHEMA>RADLEX</CODE_SCHEMA>
        <CODE_MEANING>Identificador del Paciente</CODE_MEANING>
      </CONCEPT_NAME>
      <VALUE>1</VALUE>
    </TEXT>
    <CONTAINER>
      <CONCEPT_NAME>
        <CODE_VALUE>RID29896</CODE_VALUE>
        <CODE_SCHEMA>RADLEX</CODE_SCHEMA>
        <CODE_MEANING>Mama Derecha</CODE_MEANING>
      </CONCEPT_NAME>
      <CHILDS>
        <CONTAINER>
          <CONCEPT_NAME>
            <CODE_VALUE>TRMM0009</CODE_VALUE>
            <CODE_SCHEMA>TRENCADIS_MAMO</CODE_SCHEMA>
            <CODE_MEANING>Masa</CODE_MEANING>
          </CONCEPT_NAME>
          <CHILDS>
```

```
            <TEXT>
              <CONCEPT_NAME>
                <CODE_VALUE>118522005</CODE_VALUE>
                <CODE_SCHEMA>SNOMED-CT</CODE_SCHEMA>
                <CODE_MEANING>Identificador</CODE_MEANING>
              </CONCEPT_NAME>
              <VALUE>LESION_01</VALUE>
            </TEXT>
            <NUM>
              <CONCEPT_NAME>
                <CODE_VALUE>RID29929</CODE_VALUE>
                <CODE_SCHEMA>RADLEX</CODE_SCHEMA>
                <CODE_MEANING>Cuadrante Supero-Externo (CSE) de la
Mama Derecha</CODE_MEANING>
              </CONCEPT_NAME>
              <UNIT_MEASUREMENT>
                <CODE_VALUE>000000001</CODE_VALUE>
                <CODE_SCHEMA>UNIT_MEASUREMENT</CODE_SCHEMA>
                <CODE_MEANING>Unidades Boleanas</CODE_MEANING>
              </UNIT_MEASUREMENT>
              <VALUE>1</VALUE>
            </NUM>
  …
```

Figura 45 – Fragmento de fichero XML de una instancia de una exploración mamaria.

La cabecera del fragmento de la figura 45 en la propuesta de grafo sería la siguiente:

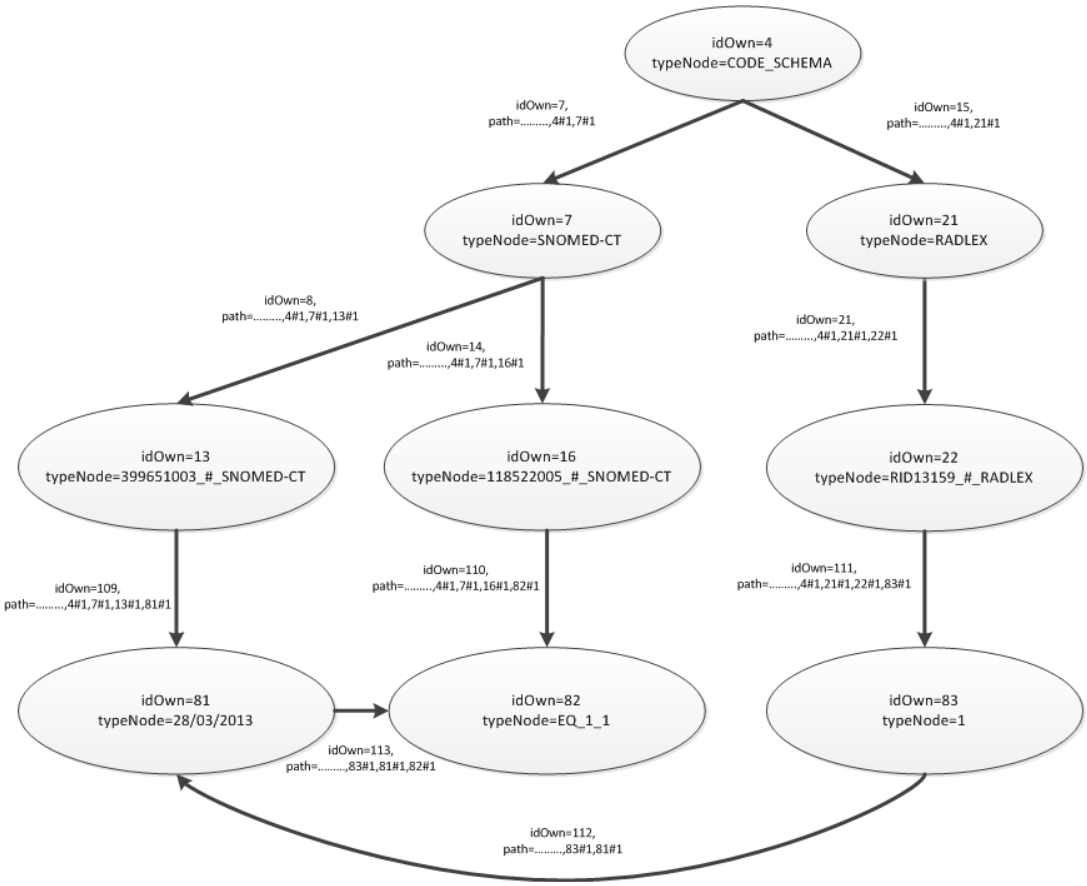

Figura 46 – Grafo de representación de cabecera de instancia de ontología.

En principio vamos a localizar el valor del “”Cuadrante Supero-Externo (CSE) de la Mama Derecha”” de un paciente, que en una fecha se le realizó una exploración de la mama derecha en la que aparece una masa. Para indicar en el modelo que el “Cuadrante Supero-Externo (CSE) de la Mama Derecha” es la localización de la masa se asignará el valor de **“1”** o **“True”**, creando (o reutilizando) un nodo con dicho valor, y relacionándolo con el nodo con valor **“RID29929_#_RADLEX”**.

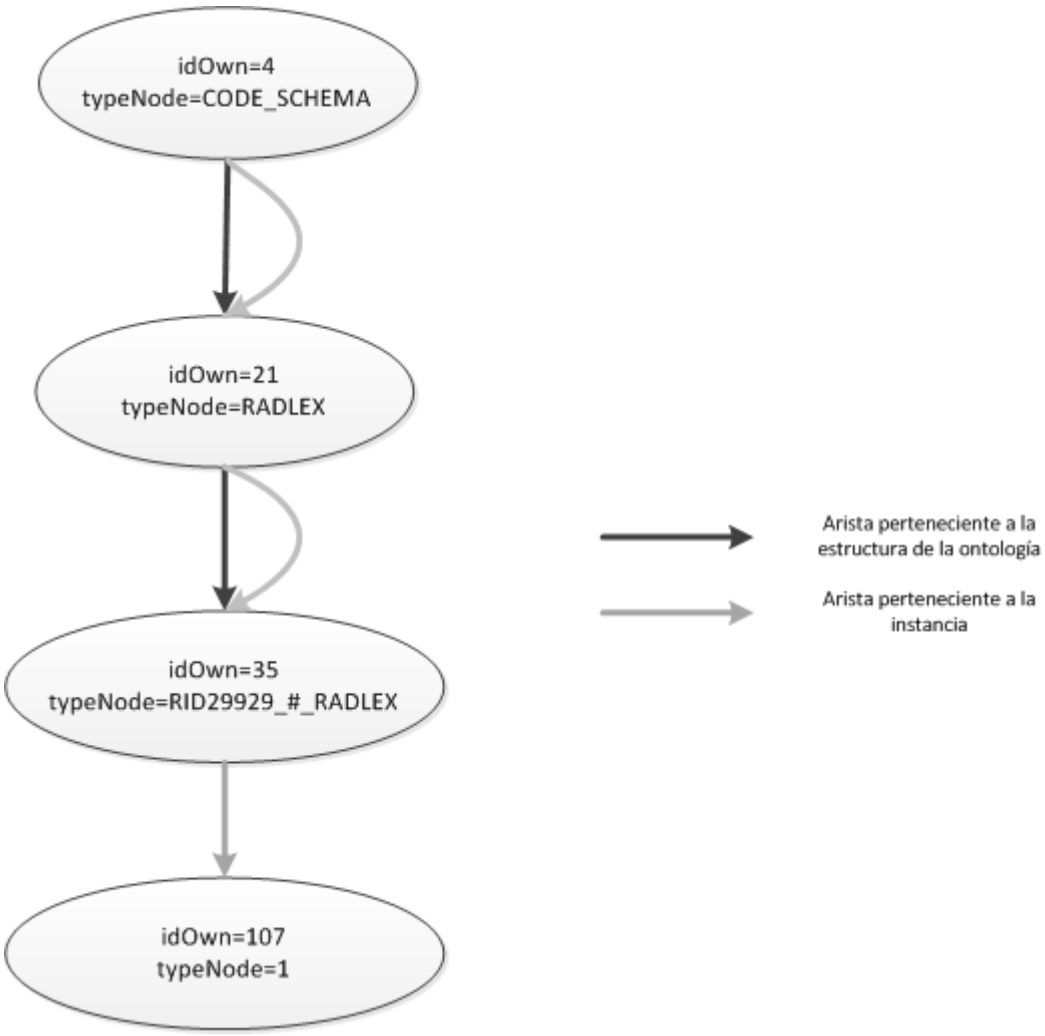

Figura 47 – Representación de valor de un elemento en el grafo.

4.1.1.4. Ventajas del modelo 1.0.

El modelo presentado aporta ventajas sobre los modelos utilizados actualmente en TRENCADIS (AMGA en este caso es el más usado), así como la organización de las estructuras de los informes en el mismo:

- **Flexibilidad**: AMGA es un modelo poco flexible, de forma que realizar operaciones que modifiquen la estructura de la ontología son complicadas de

hacer. Agregar un campo a la estructura no resulta problemático, en el sentido de crear una nueva estructura dentro del árbol, pero cambiar o actualizar un campo implica una reestructuración de las colecciones (o crear un nuevo árbol desde cero por separado), con lo que el coste de la operación es muy alto. En nuestro modelo el coste de la operación de añadir una nueva estructura es mínimo (será comentado a continuación).

- **Relaciones entre ontologías**: AMGA representa las ontologías en árboles por separado, no siendo posible correlacionar términos entre ellos mismos e integrarlos en una única estructura (sin tener que realizar una fusión). En cambio, en la propuesta realizada permite la fusión de grafos con lo que desde el primer momento todas las estructuras de informes se fusionan al darlas de alta, por lo que las relaciones entre ellos se producen de forma natural.

Otra de las ventajas que nos va a proporcionar el modelo propuesto es la continua reutilización de los nodos, ya que, conforme se van dando de alta estructuras o instancias en el sistema, se van reutilizando los nodos existentes.

Esta reutilización se realiza mediante una operación de fusión de grafos. En concreto podemos encontrar dos situaciones:

- Fusión de grafos por la creación en el grafo general de una estructura de tipo estructura de informe estructurado (también denominado **ontología**).
- Fusión de grafos por la creación en el grafo de una estructura de tipo instancia de informe estructurado.

Tanto en el caso de la creación de una nueva estructura de tipo ontología, como de tipo instancia, se creará en memoria el subgrafo necesario, de forma que una vez que el subgrafo está finalizado pasa a un proceso de fusión con la información contenida en la base de datos de grafos. En la figura 48 podemos ver el ejemplo de dos grafos que se van a fusionar, mientras que en la figura 49 vemos el resultado de la operación. En este caso se trata de dos grafos genéricos usados para representar la operación, sin el atributo *typeNode*, pero con el *path* en las aristas.

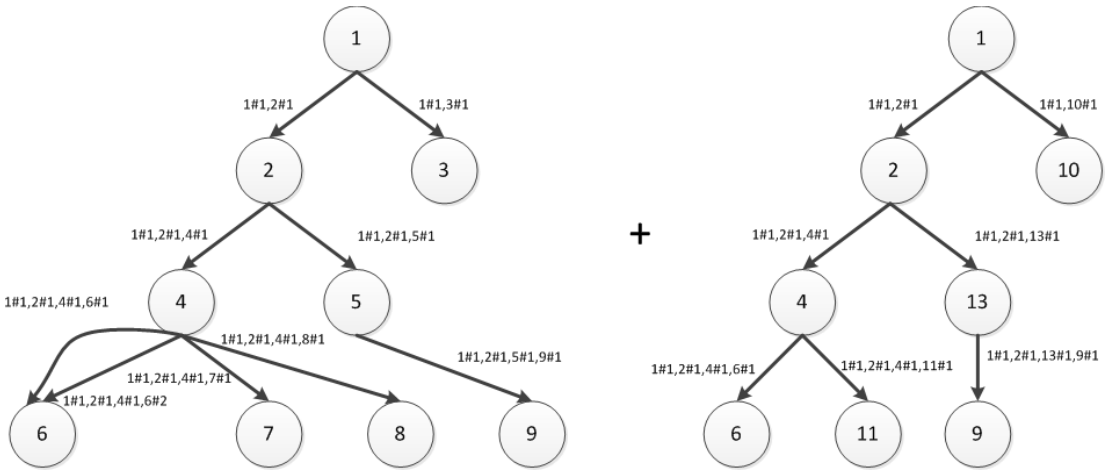

Figura 48 – Ejemplos de grafos para fusionar.

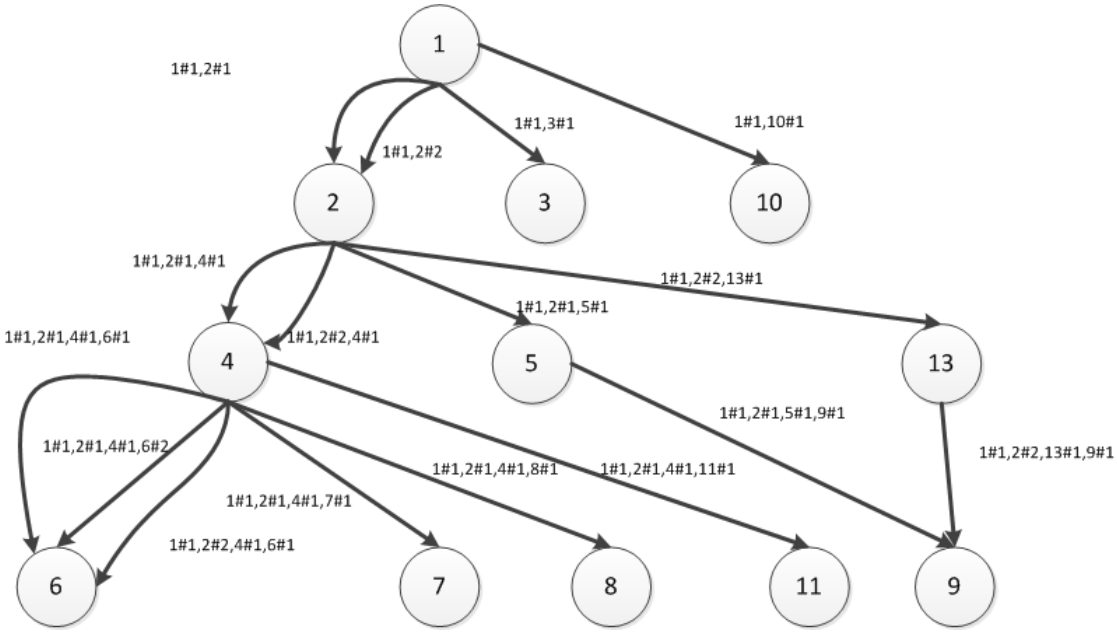

Figura 49 – Grafos de la figura 48 fusionados.

Otro de los beneficios que va a permitir este modelo es una gran flexibilidad en su estructura, de forma que es posible añadir o eliminar elementos dentro de una ontología (o instancia) con las simples operaciones de crear o eliminar aristas (siempre que los nodos ya existan), y teniendo en cuenta que se ha de reconstruir de forma correcta la propiedad *path*.

4.1.2. Desarrollo e implementación.

4.1.2.1. Descripción de los módulos.

Para el desarrollo del modelo se ha creado una serie de módulos que serán los necesarios para poder acceder a la base de datos de Neo4j, manteniendo una interfaz que sigue el estándar propuesto en el *TRENCADIS Java API Indexer* actual del *middleware* TRENCADIS, de forma que añadirá un nuevo *Indexer* al servicio propuesto (actualmente se está usando AMGA) y se creará una nueva API.

Para el desarrollo de los módulos se ha utilizado:

- Oracle Java 7.
- Neo4j 1.8.3 Stable.

Se ha optado por usar Oracle Java 7 debido a que a partir de febrero del 2013 Oracle ha finalizado el soporte a largo plazo de la misma. Esto hace que los desarrolladores de Neo4j hayan optado por usar la distribución de Java 7 para sus desarrollos, por lo que para mantener la compatibilidad usaremos dicha versión.

Para empezar recordaremos el estado actual en la figura 9, donde el componente *TRENCADIS Java API Indexer* se encuentra implementado originariamente..

La modificación realizada propone crear una nueva implementación de *Indexer*, de forma que el componente *TRENCADIS Java API Indexer* podrá optar a una de las diversas implementaciones.

Se ha definido una interfaz denominada *IDSRService* que proporciona las cabeceras de los métodos que se han de implementar por parte de las distintas APIs. La clase que proporciona la implementación de los métodos se denomina *DSRNeo4jHelper* (al igual que la que implementa AMGA que se denomina *DSRAMGAHelper*).

En el trabajo de [32] la arquitectura fue definida de forma que fueron implementados los distintos servicios de AMGA y LFC en el mismo módulo, a raíz de desplegar los servicios en distintas máquinas virtuales, y poder cambiar diversos paquetes de dichos servicios (por ejemplo cambiar el despliegue de AMGA por un despliegue en Neo4j) se separaron las operaciones de AMGA y LFC, de forma que el módulo implementado en AMGA, y por consiguiente el modulo implementado en Neo4j, no opera en ningún momento con LFC.

Al igual que en [32] la entrada de las operaciones de añadir estructura o instancia serán ficheros XML de estructura o de instancias, donde los ficheros de estructura y de instancias tendrán la extensión *.xml*. Las operaciones de borrar estructura o borrar instancia necesitarán simplemente del identificador de la estructura (ontología) o de una instancia (necesitará también del identificador de estructura).

Los ficheros de entrada han de ser válidos para la realización de las distintas operaciones. Esto es debido a que para poder tratar dichos ficheros se ha implementado una serie de *parsers SAX* para las distintas operaciones. En todo caso, si se produce un error se producirá una excepción y se cancelará la operación (es necesario recordar que las operaciones se introducen en transacciones).

Los *parsers SAX* de las operaciones de añadir estructura o instancia construirán en memoria tanto los nodos como las aristas, para poder hacer a continuación un volcado de memoria de la estructura a Neo4j en una transacción, fusionándolo así con el grafo general.

Los paquetes diseñados para el componente *Indexer* son los siguientes:

- *trencadis.infrastructure.services.dicomstorage.indexer.supports.neo4j.dsr*: paquete que contiene el módulo *DSRNeo4jHelper*, siendo la API principal que implementa la interface *IDSRService*.
- *trencadis.infrastructure.services.dicomstorage.indexer.supports.neo4j.dsr.parsers*: paquete que contiene los distintos *parsers SAX* acerca de las operaciones de añadir estructura o añadir instancia, con la finalidad de construir en memoria los grafos necesarios. Contiene 2 clases, también denominados *EdgeSax* y *NodeSax*, que hacen referencia a clases de envoltura necesarias (*wrappers*) para la creación de aristas y nodos en memoria.

- *trencadis.infrastructure.services.dicomstorage.indexer.supports.neo4j.dsr.wrapper*: paquete que contiene las clases que envuelven los distintos nodos y aristas para el paso desde memoria a base de datos Neo4j.
- *trencadis.infrastructure.services.dicomstorage.indexer.supports.neo4j.dsr.servicedb*: paquete que contiene el módulo de acceso a la base de datos (denominado *ServiceDBNeo4j*), que es tratado en este caso como un objeto embebido dentro del módulo *DSRNeo4jHelper*. Además de ello, la creación y gestión de la base de datos no depende de un SGBD externo, sino que esta embebido en el módulo *ServiceDBNeo4j*, por lo que, durante la creación del objeto del tipo *ServiceDBNeo4j*, contenido dentro del módulo *DSRNeo4jHelper* se crea y se configura la base de datos en Neo4j.

Las operaciones descritas en el interfaz *IDSRService* que implementa el módulo *DSRNeo4jHelper* son las siguientes:

- **Añadir estructura (*addStructure*)**: crea un árbol, formado por nodos y aristas, en Neo4j, y mantiene la estructura de la ontología de entrada. Para realizar dicha operación construye en memoria el árbol para luego volcarlo a la base de datos en una transacción. Reutilizará los nodos ya existentes creando solamente las aristas necesarias, por lo que todas las ontologías se fusionarán en un solo grafo conforme se vayan dando de alta.
- **Borrar estructura (*removeStructure*)**: borra la estructura de árbol perteneciente a una ontología, junto con las instancias de la misma. Los nodos no los borra debido a la reutilización de los mismos por otras ontologías y sus instancias, a excepción de aquellos nodos que se aíslen, de forma que no tienen ninguna arista. Para el borrado, únicamente, es necesario el identificador de la ontología.
- **Añadir instancia (*addReport*)**: añade una nueva instancia al grafo, relacionándolo a la estructura de un informe. Se reutilizan los nodos del grafo existentes, y se crean los nodos con los valores propios que no existan.
- **Borrar instancia (*removeReport*)**: borra las aristas pertenecientes a una instancia de una ontología. Si alguno de los nodos es aislado de forma que no tiene ninguna arista es borrado. Para realizar la operación es necesario el identificador de la ontología, así como el identificador de la instancia.
- **Listar instancia (*listReport*)**: lista la información de una instancia en concreto de un informe estructura en particular. Para realizar la operación es necesario el identificador de la ontología, así como el identificador de la instancia.

4.1.2.2. Descripción de los servicios.

Hasta este momento se ha descrito la funcionalidad de los módulos, pero no se ha comentado la integración. En [32] fue modificado los servicios de *Storage DICOM*, *Storage Broker* y *Ontologies Server* para adaptarlos al catálogo de AMGA. En nuestro caso es necesario volver a adaptarlos ya que algunos han de cambiar para poder trabajar de forma correcta con Neo4j.

La inicialización de los distintos servicios, así como la descripción de los distintos métodos publicados se encuentra en [32], por lo que no se ve necesario volver a describirlos de nuevo. Si comentaremos la función de los tres servicios en el sistema, así como si han sufrido cambios o no.

Servicio DICOM Storage.

La finalidad del servicio es gestionar las instancias de los DICOM-SR en cualquiera de los repositorios, ya sea AMGA o Neo4j. El problema surge con el tipo de implementación actual del servicio, ya que actualmente soporta la posibilidad de crear múltiples recursos para AMGA, de forma que asigna a cada ontología un recurso propio.

En nuestro caso todas las ontologías están soportadas sobre el mismo recurso, con la finalidad de poder relacionar toda la información y así buscar nuevas relaciones entre distintas instancias de distintas ontologías.

Para ello es necesario acudir a un servicio Grid de tipo *singleton*, de forma que solo exista una instancia de dicho servicio. Esto también es debido a que la base de datos se encuentra embebida (como ya se ha comentado con anterioridad) en el módulo *ServiceDBNeo4j*, de forma que dicho módulo controla el acceso a la misma.

El servicio *DICOM Storage* ha de permitir que sea *singleton* (en el caso de Neo4j), y de múltiples recursos (en el caso de AMGA).

Servicio Storage Broker.

El servicio *Storage Broker*, implementado actualmente para AMGA, crea recursos que actúan como mediadores para cada tipo de ontología, permitiendo consultas simultáneas en todos los repositorios AMGA del sistema. Al existir un único repositorio no será necesario mediar por cada tipo de ontología, y será el propio gestor de bases de datos el que controle las consultas simultáneas.

Servicio Ontologies Server.

El servicio *Ontologies Server* se encarga de almacenar las ontologías de los informes estructurados, así como de mantener actualizado el servicio del *DICOM Storage* y el *Storage Broker*.

Actualmente se dedica a almacenar las ontologías y no a actualizar el *DICOM Storage* ni el *Storage Broker*, debido principalmente a que un cambio en una estructura de una ontología es una operación crítica.

4.1.3. Pruebas realizadas sobre el modelo 1.0.

Tras implementar las distintas clases que realizan la funcionalidad de las distintas operaciones sobre la base de datos Neo4j se procede a realizar un conjunto de pruebas para verificar la viabilidad del modelo.

A partir de esto se han realizado 2 tipos de experimentos dentro de esta propuesta:

- Hemos realizado un conjunto de inserciones de informes estructurados para distintos tamaños del problema, que van desde 50 valores dentro del fichero XML a 500 (tendrán 50 nodos valor, 100 nodos valor,...hasta 500 nodos valor). Para cada tamaño del problema se han insertado 10 nuevos datos, obteniendo el tiempo medio de inserción para cada tamaño específico.
- La resolución de 5 consultas, obteniendo de la misma forma su tiempo medio.

La base de datos se encuentra poblada con una única estructura que representa la exploración mamaria, y con 9.000 instancias (exploraciones) de 1.000 pacientes tomadas en los primeros 6 meses del 2013, por lo que, en términos de grafos, nos situamos en 19.346 nodos y 3.722.110 relaciones existentes en la base de datos.

Para las operaciones de inserción hemos lanzado un proceso que calculará el tiempo medio para los distintos tamaños de nodos nuevos a crear en la base de datos de Neo4j. Los resultados obtenidos son:

Número de campos	Tiempo de inserción modelo 1.0 en Neo4j	Tiempo de inserción en AMGA
50	0,475 s	0,589 s
100	0,506 s	0,702 s
150	0,467 s	0,801 s
200	0,570 s	1,281 s
250	0,636 s	1,423 s
300	0,598 s	1,784 s
350	0,687 s	2,635 s
400	0,871 s	3,070 s
450	0,775 s	3,304 s
500	0,852 s	3,636 s

Tabla 1 - Tiempos de la operación de insertar en las distintas implementaciones.

Es necesario destacar que el proceso de inserción en la base de datos no es, en absoluto similar en Neo4j y en AMGA. En ambos existirá un parseador que construirá la instancia en memoria, y a continuación se procederá a la inserción de los datos. Debido a las características propias de los grafos, Neo4j reutilizará todos los nodos ya existentes, por lo que pensar en la creación de n nuevos nodos en una instancia puede implicar que el fichero XML de la instancia puede llegar a contener un gran número de valores (lesiones, características, valores propios, etc.).

En cuanto a las consultas a realizar son:

1. Obtener el número de todas las exploraciones mamarias realizadas.
2. Obtener el identificador de todos los informes que se le han realizado al paciente 461.
3. Obtener el identificador de todos los informes que se le han realizado al paciente 461 con fecha posterior al 01/01/2013.
4. Obtener los identificadores de todos los informes que se la han realizado al paciente 461, con fecha posterior al 01/01/2013 y que tenga en la mama derecha lesiones de tipo "masa" en la localización "retroareolar".
5. Obtener la fecha y el número de exploraciones mamarias que se le han realizado al paciente 461.

La solución a las distintas consultas (en *Cypher*, el lenguaje de Neo4j) son las siguientes:

- Consulta 1:

```
START n=node:nodes_typeNode(typeNode='DICOM_SR')
MATCH n-[f]->m
WHERE n.IDOntology='4' AND
m.typeNode='DICOM_SR_DATA'
RETURN count(m);
```

- Consulta 2:

```
start a=node:nodes_typeNode('typeNode:DICOM_SR'),
c=node:nodes_typeNode('typeNode:461')
match a-[r]->b-[s]->c
where  a.IDOntology='4' and
b.typeNode='DICOM_SR_DATA' and
r.register=s.register and
s.path=~ r.path + '.*'
with c,s
match c-[t]->d-[u]->e
where t.register=s.register and
u.register=t.register and
t.path=~ s.path + '.*' and
u.path=~ t.path + '.*'
return e.typeNode;
```

- Consulta 3:

```
start a=node:nodes_typeNode('typeNode:DICOM_SR'),
c=node:nodes_typeNode('typeNode:461')
match a-[r]->b-[s]->c
where a.IDOntology='4' and
b.typeNode='DICOM_SR_DATA' and
r.register=s.register and
```

```
s.path=~ r.path + '.*'
with c,s
match c-[t]->d-[u]->e
where t.register=s.register and
t.path=~ s.path + '.*' and
u.register=t.register and
u.path=~ t.path + '.*' and
d.typeNode>='20130101'
return e.typeNode;
```

- Consulta 4:

```
start a=node:nodes_typeNode('typeNode:DICOM_SR'),
c=node:nodes_typeNode('typeNode:461'),
a3=node:nodes_typeNode('typeNode:RID29896_#_RADLEX'),
a4=node:nodes_typeNode('typeNode:TRMM0009_#_TRENCADIS_MAMO'),
a7=node:nodes_typeNode('typeNode:RID29950_#_RADLEX')
match a-[r]->b-[s]->c
where  a.IDOntology='4' and
b.typeNode='DICOM_SR_DATA' and
r.register=s.register and
s.path=~ r.path + '.*'
with c,s,a3,a4,a7
match c-[t]->d-[u]->e
where t.register=s.register and
u.register=t.register and
t.path=~ s.path + '.*' and
u.path=~ t.path + '.*' and
d.typeNode>='20130101'
with a3,a4,u,a7,e
match a3-[r1]->a5
where r1.register=u.register and
r1.path=~ r1.path + '.*'
with a4,r1,a7,e
match a4-[r2]->a6
where r2.register=r1.register and
r2.path=~ r1.path + '.*'
with a7,r2,e
match a7-[r3]->a8
where r3.register=r2.register and
r3.path=~ r2.path + '.*' and
a8.typeNode='1'
return distinct(e.typeNode);
```

- Consulta 5:

```
Start a=node:nodes_typeNode('typeNode:DICOM_SR'),
c=node:nodes_typeNode('typeNode:461')
match a-[r]->b-[s]->c
```

```
where  a.IDOntology='4' and
b.typeNode='DICOM_SR_DATA' and
s.path=~ r.path + '.*'
with c,r
match c-[t]->d-[u]->e
where t.path=~ r.path + '.*' and
u.path=~ t.path + '.*'
return d.typeNode As FechaExploracion, count(e) as
Exploraciones;
```

Los resultados obtenidos de las ejecuciones de estas consultas son los siguientes (la consulta 5 no es posible realizarla directamente en AMGA en una única instrucción):

Consulta	Tiempo de búsqueda modelo 1.0 en Neo4J	Tiempo de búsqueda en AMGA
1	0,91 s	0,055 s
2	0,93 s	0,060 s
3	1,02 s	0,064 s
4	4,31 s	2,344 s
5	1,045 s	---

Tabla 2 - Tiempos de la operación de búsqueda en las distintas implementaciones.

El resultado obtenido con las consultas no es el esperado, ya que la profundidad de los elementos seleccionados en las consultas ha afectado el tiempo de respuesta de las mismas. En el caso del nuevo modelo dicho tiempo ha crecido.

En cada implementación la situación es distinta: en caso de AMGA se procede a descender en la estructura del árbol de directorios, mientras que en el modelo de grafos se procede a dar saltos entre nodos. En principio estos resultados no deberían ser así.

Tras analizar los resultados y el modelo, se han encontrado una serie de nodos que actúan como enlaces entre las ontologías y las instancias, que sufren una sobrecarga de aristas que, aunque son deseables desde un punto de vista médico para descubrir relaciones deterministas o probabilistas entre la información disponible, proporciona un costo adicional de latencia.

4.1.4. Conclusiones.

El modelo presentado en esta primera aproximación no cumple con las expectativas esperadas. Aunque la inserción es más rápida que el modelo en AMGA, la exploración mediante búsquedas en la base de datos en Neo4j no es eficiente.

La reutilización de diversos nodos en el grafo produce una sobrecarga en los mismos que añade una latencia a las distintas consultas. Esto hace que las consultas pasen por una serie de aristas y nodos innecesarios creados, principalmente, para el mantenimiento de la jerarquía de 3 niveles.

El resultado obtenido puso de manifiesto que la flexibilidad y reutilización, aunque son ventajas muy grandes, no son convenientes para dicho modelo, por lo que se pasó a refinar el modelo en otro.

4.2. Modelo V.2.0: Representación simplificada de las estructuras de los informes estructurados y de las instancias de las mismas en un grafo.

4.2.1. Estudio, análisis y diseño del modelo versión 2.0.

Tras las pruebas realizadas en el primer modelo creado se ha llegado a una serie de conclusiones previas:

- La reutilización de los nodos implica una sobrecarga de tiempo a la hora de implementar las búsquedas en el grafo.
- La creación del atributo *path* para almacenar los distintos caminos de cada arista implica que, a medida que el nivel de profundidad aumenta, aumenta enormemente el valor de dicho atributo, siendo muy costoso realizar comparaciones sobre el mismo.
- El proceso de generación en memoria de toda la estructura de árbol para a continuación volcarlo en Neo4j implica hacer uso de dos *wrappers* para los distintos nodos y aristas (uno para memoria y otro para Neo4j), generando una ralentización en la operación.
- En cada operación de inserción de instancias se realizan búsquedas continuas de los nodos pertenecientes a la estructura de la ontología, de forma que son operaciones muy repetitivas.
- El modelo obliga a pasar por nodos que aportan poca información a la consulta realizada, creando más profundidad en las distintas consultas y obligando al sistema a recorrer diversas veces la jerarquía de 3 niveles hasta llegar al nodo deseado.

Con estas conclusiones se ha diseñado un nuevo modelo que permita solventar los problemas descritos, a costa de perder ciertos beneficios que disponíamos con el anterior modelo presentado.

El modelo versión 1.0 se basa en el volcado completo de los ficheros XML al grafo, creando un modelo de 3 niveles de jerarquía para darle un sentido semántico a la

información, y de esta forma, que las instancias almacenadas en el mismo tengan un significado.

En este nuevo modelo los 3 niveles de jerarquía desaparecen, ya que no se produce un volcado completo del fichero XML, sino un volcado parcial, creando únicamente los nodos necesarios para poder mantener la información de las instancias.

Los únicos nodos necesarios que participarán en el nuevo modelo son los nodos que identifican un elemento dentro del fichero XML, o sea el *CODE_VALUE* y el *CODE_SCHEMA*, y en dichos nodos seguiremos manteniendo los atributos *idOwn* para identificar al nodo y *typeNode* para almacenar el identificador (o valor en los campos de las instancias). Siguiendo el mismo uso que se le dio en el anterior modelo, podemos usar la unión de ambos para crear un nodo que identifique un elemento dentro del informe estructurado, así como los distintos valores asociados a la instancia.

La idea de este nuevo modelo es simplificar lo máximo posible la creación del grafo, por lo que no vamos a acudir al principio de reutilización de nodos usado en el primer modelo. Esto va a hacer que, a priori, nuestro grafo ocupe más espacio en disco, ya que, al no usarse dicho principio todas las etiquetas del fichero XML deberían convertirse en un nuevo nodo, pero no es así. Ya hemos comentado que este nuevo modelo se basaba en simplificar el contenido del fichero, de forma que solamente crearemos nodos identificadores que contengan el *CODE_VALUE* y el *CODE_SCHEMA* unidos por "_#_".

Pongamos como ejemplo la cabecera de la plantilla del informe estructurado de la exploración de mama de la figura 45, donde se creaban todos los nodos necesarios para implementar todos los elementos del fichero XML. En este caso simplificaremos, de forma que la cabecera sería de la siguiente forma:

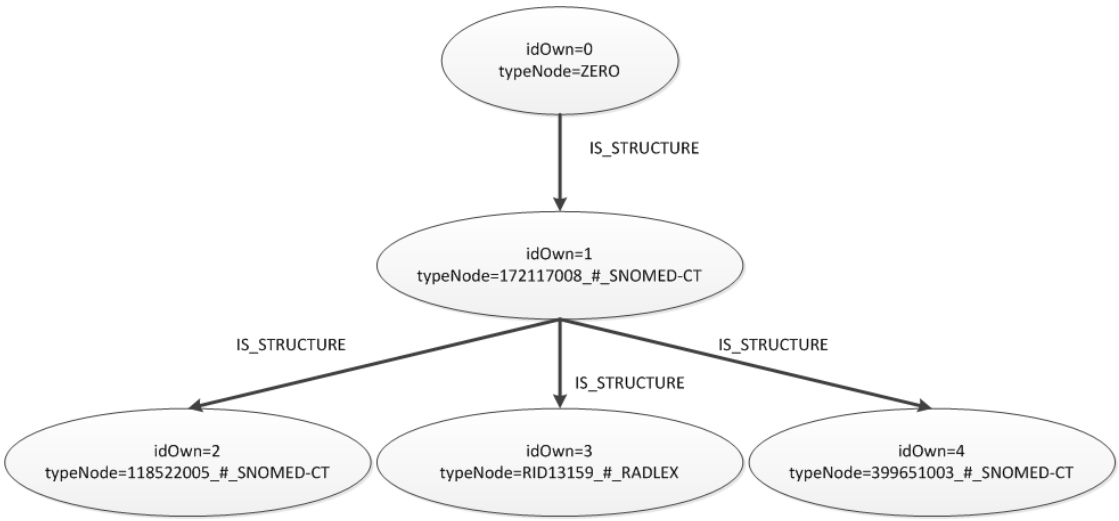

Figura 50 - Grafo simplificado de la cabecera de la ontología.

La figura 50 indica que solo existen en el grafo los nodos identificadores necesarios para mantener la estructura coherente de la cabecera. Al crear esta estructura en forma de árbol no es necesario acudir a los 3 niveles de jerarquía definidos en el modelo 1 para dotar al grafo de sentido semántico.

En cuanto a las aristas, las distintas propiedades que las acompañan desaparecen ya que no son necesarias. Por conveniencia a la hora de usar las consultas, sí se han definido que se producen dos tipos de relaciones distintas dentro del grafo:

- Relaciones de estructura: estas relaciones se producen única y exclusivamente entre nodos que son creados al dar de alta una ontología en el sistema. Estos nodos pertenecen al grupo de nodos estructurales.
- Relaciones de valor: estas relaciones se producen entre un nodo identificador que pertenece a la estructura de una ontología y un nodo que corresponde a un valor del elemento de la instancia (que corresponde al nodo identificador seleccionado).

Tras crear los nodos cabecera (siguiendo el ejemplo anterior) podemos crear los elementos que apuntan a la mama derecha (que corresponde al nodo con valor "RID29896_#_RADLEX") y a la mama izquierda (que corresponde al nodo con valor "RID29897_#_RADLEX").

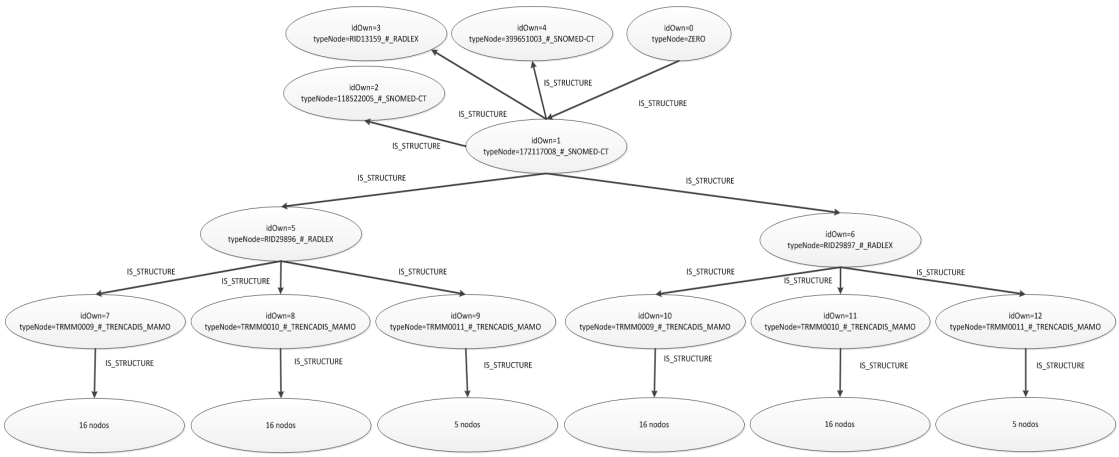

Figura 51 - Representación de la mama derecha y la mama izquierda dentro de la ontología.

Se puede comprobar en la figura 51 que existen nodos repetidos, tanto los correspondientes a la mama derecha, como a la mama izquierda. Esto permite no sobrecargar los distintos nodos con aristas, de forma que se obtendrán mejores resultados en las consultas.

Los últimos nodos de la estructura mantendrán los valores de las distintas instancias, que suelen ser los nodos más profundos de la jerarquía en los ficheros XML. Un ejemplo de dichos nodos pueden ser los siguientes (hay que recordar que se repiten):

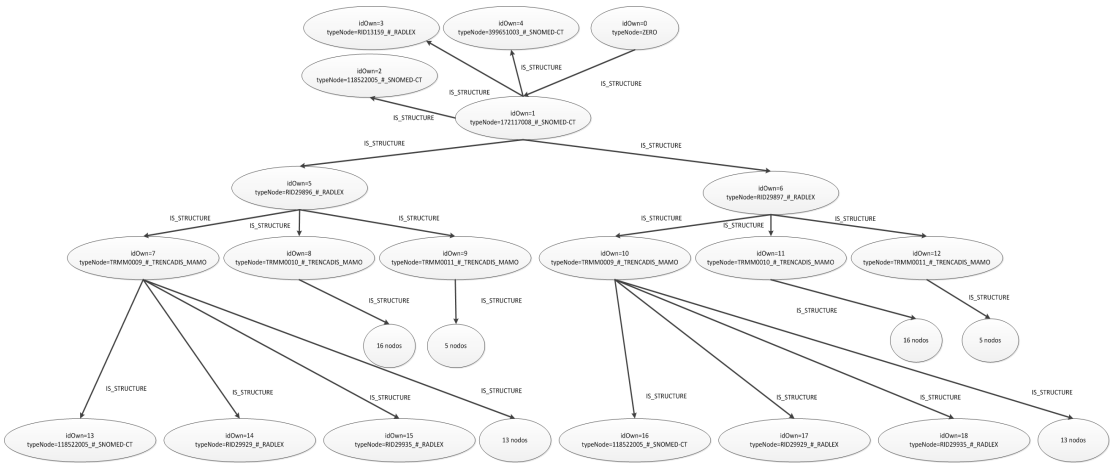

Figura 52 – Fragmento de representación de masa situado en la mama derecha y en la izquierda.

El árbol (o grafo) que se crea a partir del XML es mucho más sencillo que el que se crea en la versión 1.0, donde no se reutilizan los nodos, por lo que no es tan flexible como el modelo anterior.

La no reutilización de los nodos obliga a crear una estructura nueva por cada ontología, de forma que todas las ontologías comparten un único nodo común, el nodo *ZERO* (a excepción del nodo *ZERO*, es similar al modelo propuesto en AMGA). Con esta nueva situación podríamos dejar la propuesta inicial acerca del servicio *Storage Broker*, y dejarlo como un servicio con una implementación de múltiples recursos, de forma que cada una de las ontologías se encontrará en un recurso en forma de grafo.

En cuanto a la operación de insertar instancias, el proceso es algo distinto al de dar de alta ontologías. Para empezar hemos de distinguir dos fases durante el alta de las instancias: la creación de la cabecera, por un lado, y la asignación de valores a los distintos nodos identificadores, por otro.

4.2.1.1. Creación de cabecera de datos de la instancia.

Al igual que en el modelo 1.0, la cabecera tiene una construcción especial, debido a que necesitamos que los datos lleven un orden concreto para poder implementar las consultas de una forma más eficiente. La cabecera de datos tendrá el siguiente formato:

Figura 53 – Estructura de la cabecera de la instancia.

Esta construcción es imprescindible en todas las instancias de cualquier ontología. Para ello seguiremos un patrón a la hora de insertar las instancias, donde cada nodo almacenará el identificador del nodo del que tiene una dependencia. En el caso de la cabecera la dependencia se encuentra en la figura 54. En ella podemos comprobar las

dos instancias que existen, y cómo se almacena en la propiedad *registerNode* (en la figura lo podemos visualizar como *regNod*) el identificador del nodo del que depende (en dicha propiedad se almacena la dependencia).

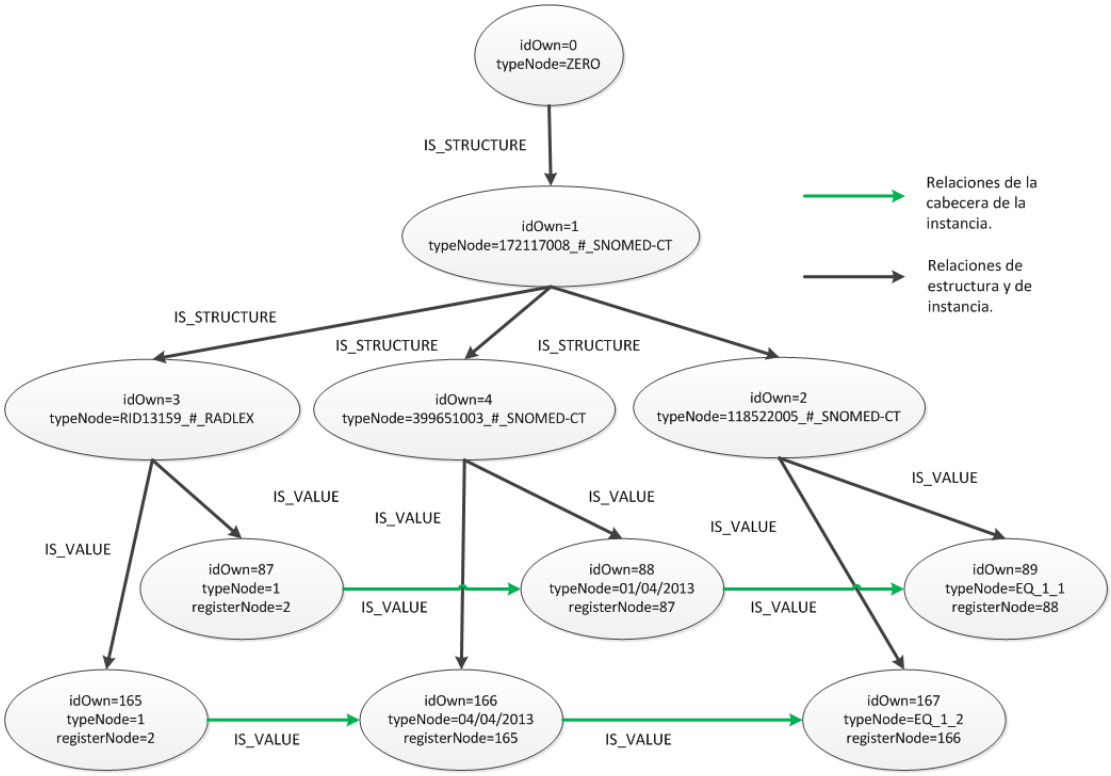

Figura 54 – Dependencia entre los nodos de instancias.

El identificador del último nodo de la cabecera, o sea el que identifica a la instancia, será el identificador maestro dentro de la instancia. De esta forma, dicho identificador permitirá crear la dependencia entre la cabecera de la instancia y los identificadores de los distintos elementos característicos dentro de la instancia (por ejemplo una masa).

4.2.1.2. Asignación de valores a los nodos identificadores.

Para la asignación de valores vamos a escoger el siguiente ejemplo:

```
<CONTAINER>
  <CONCEPT_NAME>
  <CODE_VALUE>TRMM0009</CODE_VALUE>
  <CODE_SCHEMA>TRENCADIS_MAMO</CODE_SCHEMA>
  <CODE_MEANING>Masa</CODE_MEANING>
  </CONCEPT_NAME>
  <CHILDS>
    <TEXT>
      <CONCEPT_NAME>
```

```
<CODE_VALUE>118522005</CODE_VALUE>
        <CODE_SCHEMA>SNOMED-CT</CODE_SCHEMA>
        <CODE_MEANING>Identificador</CODE_MEANING>
      </CONCEPT_NAME>
      <VALUE>LESION_01</VALUE>
    </TEXT>
    <NUM>
      <CONCEPT_NAME>
        <CODE_VALUE>RID29929</CODE_VALUE>
        <CODE_SCHEMA>RADLEX</CODE_SCHEMA>
         <CODE_MEANING>Cuadrante Supero-Externo (CSE) de la
Mama Derecha</CODE_MEANING>
       </CONCEPT_NAME>
      <UNIT_MEASUREMENT>
        <CODE_VALUE>000000001</CODE_VALUE>
        <CODE_SCHEMA>UNIT_MEASUREMENT</CODE_SCHEMA>
        <CODE_MEANING>Unidades Boleanas</CODE_MEANING>
      </UNIT_MEASUREMENT>
      <VALUE>0</VALUE>
    </NUM>
```

Figura 55 – Fragmento de instancia de informe estructurado.

En la figura 56 expondremos un ejemplo de identificador de informe estructurado, con el fin de poder crear las dependencias entre los nodos.

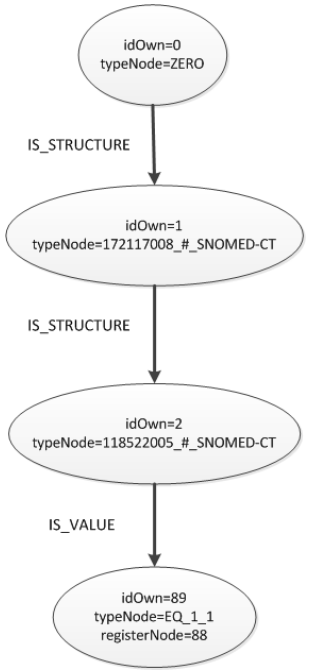

Figura 56 – Representación del identificador del informe estructurado para la figura 55.

Este proceso se encargará de crear los nodos valor necesarios y relacionarlos con los nodos identificadores, indicando en cada nuevo nodo valor su dependencia, aunque en

esta situación se identifican dos posibles casos de dependencia de los nodos que contienen los valores. Los casos de dependencia son:

- Si el elemento padre (en este caso el elemento es de tipo "masa", que está identificada por la etiqueta en amarillo) tiene un elemento hijo de tipo "Identificador" (está en verde dicha etiqueta) de entre todos sus hijos. O lo que es lo mismo, si el elemento padre contiene el valor **"118522005_#_SNOMED-CT"** entre sus elementos hijos, la dependencia de los nodos valores será la siguiente:

    - El nodo valor hijo de la etiqueta identificadora (es decir, el nodo que contendrá el valor **"LESION_01"** en azul) tendrá una dependencia del identificador maestro de la instancia. Dicho identificador maestro de la instancia se puede visualizar en la figura 56, y dicha dependencia creada en la figura 57.
    - Los nodos hijos distintos al tipo **"Identificador"** contenidos dentro del nodo padre (por ejemplo el "Cuadrante Supero-Externo (CSE) de la Mama Derecha" que se encuentra su identificación en color gris) tendrá una dependencia del identificador del nodo valor **"LESION_01"**. En la figura 58 podemos comprobar su dependencia.

- Si el elemento padre no tiene un elemento hijo de tipo "Identificador", o sea con valor **"118522005_#_SNOMED-CT"**, todos sus elementos hijos dependerán del identificador maestro de la instancia (un claro ejemplo son todos los elementos hijos que son contenidos dentro del elemento padre de tipo "Hallazgos Asociados"). En la figura 59 podemos comprobar su dependencia.

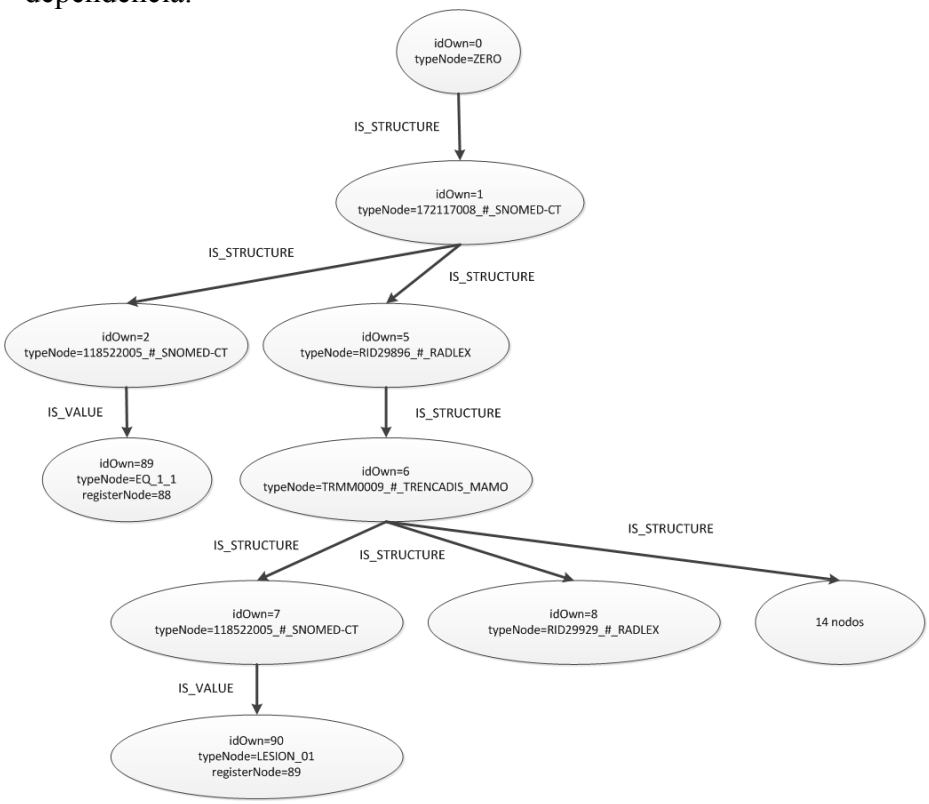

Figura 57 – Representación de identificador de masa en la mama derecha del informe estructurado de la figura 55 y figura 56.

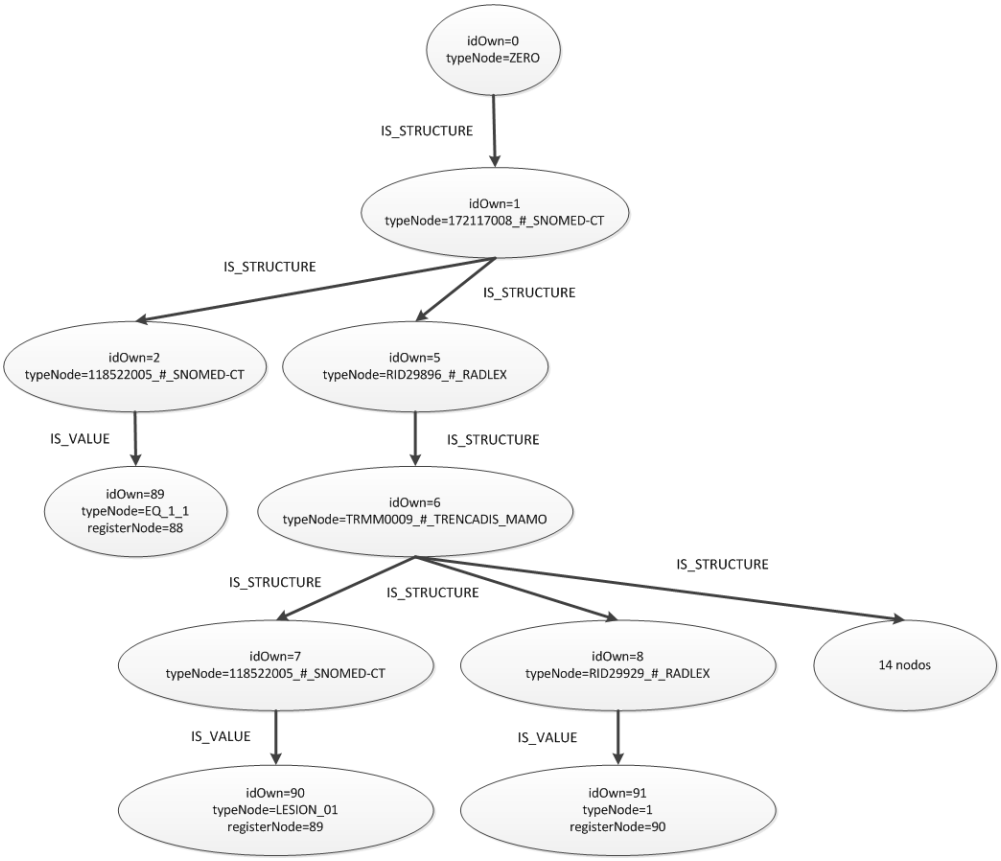

Figura 58 – Representación de valores de la masa del informe estructurado de la figura 55 y figura 56.

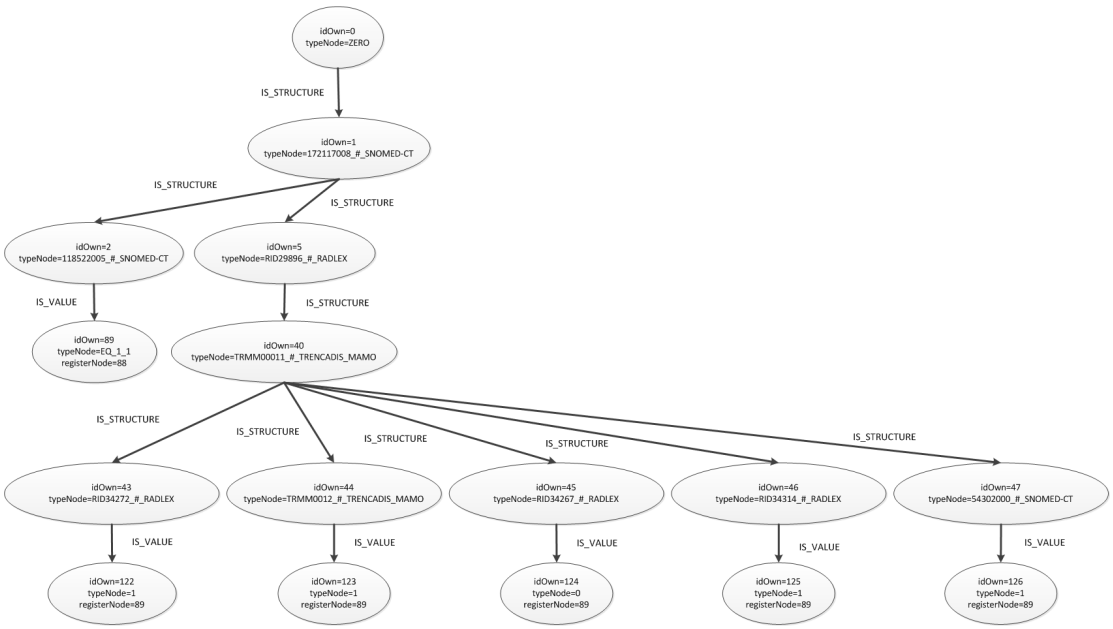

Figura 59 –Representación de valores de los hallazgos asociados del informe estructurado de la figura 55 y figura 56.

Con la propuesta realizada podemos navegar por las instancias en la base de datos de una forma muy sencilla, ya que en todo momento los distintos valores de las instancias conocen de quien dependen.

De hecho podemos comprobar en la siguiente consulta como se produce la dependencia entre los distintos nodos. La consulta sería:

"Obtener todos los identificadores de las instancias de las exploraciones mamarias realizadas al paciente con identificador 461 a partir de la fecha 2013/04/01".

Y la solución a esta consulta sería:

```
match n1-[r1:IS_VALUE]->n2:VALUE-[r2:IS_VALUE]->n3-[r3:IS_VALUE]->n4
using index n2:VALUE(typeNode)
where ID(n1)=5 and
n2.typeNode='461' and
n3.typeNode>'20130401'
with n4
match n5-[r4:IS_VALUE]->n6:VALUE
using index n6:VALUE(registerNode)
where ID(n5)=8
AND n6.registerNode=n4.idOwn
with n4,n6
match n7-[r5:IS_VALUE]->n8:VALUE
using index n8:VALUE(registerNode)
where ID(n7)=9
AND n8.registerNode=n6.idOwn
AND n8.typeNode='1'
return distinct(n4.typeNode) as Identificador_Informe;
```

Podemos comprobar en la solución cómo se produce una dependencia mediante el atributo *registerNode* que busca su equivalencia en el identificador de otro nodo.

Además de esto, se ha aprovechado una nueva mejora en Neo4j que permite crear grupos con los nodos a partir de una asignación de etiquetas a los mismos. En Neoj4 esta característica se denomina ***labels***. De esta forma se han agrupado los nodos en dos tipos de etiquetas:

- *VALUE*: Corresponden a nodos que contienen valores de las instancias.
- *STRUCTURE*: Corresponden a nodos que contienen identificadores de la estructura de una ontología.

4.2.1.3. Características de flexibilidad y reutilización de nodos en el modelo.

En este modelo simplificado la flexibilidad respecto a las operaciones de creación, modificación y eliminación se respeta. Esto es muy importante con respecto a AMGA, ya que, dichas operaciones son muy costosas dentro de la estructura jerárquica de directorios de la cual está compuesta la implementación actual. Es posible añadir un nuevo elemento a una ontología (o eliminarlo) de forma muy sencilla.

En cuanto a la característica de reutilización de nodos usado en el modelo 1 no aparece en este modelo actual, ya que, como se ha comprobado en las diversas pruebas

del modelo anterior, dicha característica introduce una latencia en el tiempo de respuesta de las consultas, de forma que la ventaja obtenida penaliza el modelo. Por ello se optó por no reutilizar los nodos a la hora de crear nuevas instancias o ontologías, y pasar así a crear nuevos nodos que se repiten, pero las cuales solo disponen de una única arista.

4.2.1.4. Relación entre las distintas ontologías.

Al estar todas las ontologías en un único grafo, como ocurre en el modelo 1, todas ellas se encuentran relacionadas entre sí, de forma que es bastante sencillo poder realizar consultas "transversales" sobre todas las ontologías. En este nuevo modelo (al igual que AMGA) las ontologías se encuentran en recursos distintos, por lo que no es posible implementar dicha transversalidad.

Esta característica importante que se encontraba en el modelo 1 se pierde en el modelo 2, por lo que es necesario, en un futuro, implementar un sistema que permita conectar las distintas ontologías entre ellas.

4.2.2. Desarrollo e implementación.

4.2.2.1. Descripción de los módulos.

Para el desarrollo del nuevo modelo hemos partido del desarrollo realizado en el modelo anterior, de forma que las operaciones siguen realizándose mediante *parsers*, y la base de datos sigue estando embebida dentro del módulo que implementa el servicio de la misma.

En este caso, para acelerar las consultas se ha implementado un módulo de ***servicio de cache*** sobre los nodos estructurales, de forma que a la hora de construir las consultas se realiza utilizando dicho servicio de cache, ya que proporciona los identificadores de los nodos estructurales de una forma muy rápida, y gracias a ellos las consultas se aceleran de una forma considerable.

Además de esto, hay que tener en cuenta que ya no se genera el grafo en memoria para luego recorrerlo e insertarlo en la base de datos, sino que se crea una transacción y se va creando el grafo según se va recorriendo el fichero XML, por lo que la operación se acelera considerablemente, tanto para la creación de una ontología, como para añadir instancias a dicha ontología.

Para el desarrollo del modelo 2.0 se ha seguido manteniendo el interfaz estándar para mantener la compatibilidad con el propuesto por TRENCADIS, aunque en este caso se ha cambiado la base de datos pasando de Neo4j 1.8.3 Stable a Neo4j 2.0.0-M04, ya que ciertos aspectos, como los *labels* comentados en el punto anterior, solamente existen a partir de dicha versión.

Los paquetes diseñados para la aplicación son los siguientes:

- *trencadis.infrastructure.services.dicomstorage.indexer.supports.neo4j.dsr*: paquete que contiene el módulo *DSRNeo4jHelper*, siendo la API principal que implementa la interface *IDSRService* (al igual que la versión 1).
- *trencadis.infrastructure.services.dicomstorage.indexer.supports.neo4j.dsr.parsers*: paquete que contiene los distintos *parsers* acerca de las operaciones de añadir estructura o añadir instancia, con la finalidad de construir en memoria los grafos necesarios.
- *trencadis.infrastructure.services.dicomstorage.indexer.supports.neo4j.dsr.wrapper*: paquete que contiene las clases que envuelven los distintos nodos y aristas.
- *trencadis.infrastructure.services.dicomstorage.indexer.supports.neo4j.dsr.servicedb*: paquete que contiene el módulo de acceso a la base de datos (denominado *ServiceDBNeo4j*), además del servicio de cache (denominado *ServiceCache*). A continuación lo vamos a tratar en un apartado aparte para explicar con detalle la nueva implementación.

Las operaciones descritas en el interfaz *IDSRService* que implementa el módulo *DSRNeo4jHelper* son las mismas descritas en el modelo 1, pero con algunas modificaciones:

- **Añadir estructura (*addStructure*)**: se encarga de crear la estructura de la ontología directamente en una transacción mediante un *parser*. No reutiliza los nodos, y conforme van dándose de alta los nodos en el grafo, son introducidos en el servicio de cache.
- **Borrar estructura (*removeStructure*)**: borra la estructura de la ontología, así como todas sus instancias.
- **Añadir instancia (*addReport*)**: añade una nueva instancia al grafo de la ontología correspondiente. No reutiliza ningún nodo correspondiente a un valor de una instancia.
- **Borrar instancia (*removeReport*)**: borra una instancia del grafo de la ontología correspondiente.
- **Listar instancia (*listReport*)**: lista la información de una instancia en concreto de una ontología en particular. Para realizar la operación es necesario el identificador de la ontología, así como el identificador de la instancia.

El componente de acceso a base de datos: *ServiceDBNeo4j*.

Sin duda se trata de la parte más importante del desarrollo:

- El acceso a la base de datos sobre Neo4j.
- Las operaciones disponibles con la base de datos en Neo4j.
- El servicio de cache (denominado *ServiceCache*) sobre los nodos estructurales de las ontologías.

El componente *ServiceDBNeo4j* se encuentra embebido en forma de objeto dentro del componente *ServiceNeo4j*. Esto permite que el acceso a dicho componente sea única y exclusivamente a través del *ServiceNeo4j*, y solo con los métodos que proporciona este último componente, coincidiendo estos con el estándar requerido por TRENCADIS.

El componente *ServiceDBNeo4j*, al crearse, realiza una apertura de la base de datos en Neo4j, o la crea si no existe. En este último caso, no solamente se crea el soporte físico en forma de ficheros de la base de datos, sino también una serie de índices que van a permitir realizar las consultas de forma más eficiente. Distinguiremos dos tipos de índices dentro de Neo4j:

- Índices sobre los grupos de etiquetas: las etiquetas (o *labels* como han sido denominadas con anterioridad) son dominios sobre los índices. Cabe recordar que en nuestro caso los dominios existentes son *STRUCTURE* para los nodos de estructura de ontología y *VALUE* para los nodos valores de las instancias. En este caso un índice es creado sobre una propiedad que tienen todos los nodos de un dominio. Un ejemplo claro sería:

  ```
  create index on :STRUCTURE(typeNode);
  ```

- Índices sobre los nodos: este índice es muy similar al anterior, pero afecta a todos los nodos, sea cual sea su dominio (si es que dispone de alguno, ya que es posible que no tenga ninguna etiqueta). Nos permite definir índices de nodos de múltiples atributos, lo que no es muy útil a la hora de consultar valores de instancias, ya que disponen de un valor (en el atributo *typeNode*), y tienen una dependencia con otro nodo (en el atributo *registerNode*). Este tipo de índices que fueron usados en el modelo 1.0, han sido sustituidos por índices sobre grupos de etiquetas.

En cuanto al componente *ServiceCache* se encuentra también embebido dentro del componente *ServiceNeo4j*. Dicho componente almacena los nodos estructurales de cada ontología, por lo que, a la hora de construir las consultas el componente proporciona el identificador de cada nodo estructural, de forma que, mediante los índices creados sobre los grupos de etiquetas, se pueden construir consultas muy eficientes.

Consultas sobre el modelo.

Las consultas en Neo4j se basan, básicamente, en localizar el nodo perteneciente a la estructura de la ontología y a continuación obtener (o filtrar) el nodo valor. Un ejemplo podría ser la siguiente consulta:

"Obtener todos los identificadores de las instancias de las exploraciones mamarias realizadas al paciente con identificador 1 partir de la fecha 01/01/2013".

En la figura 60 podremos observar la estructura de la ontología:

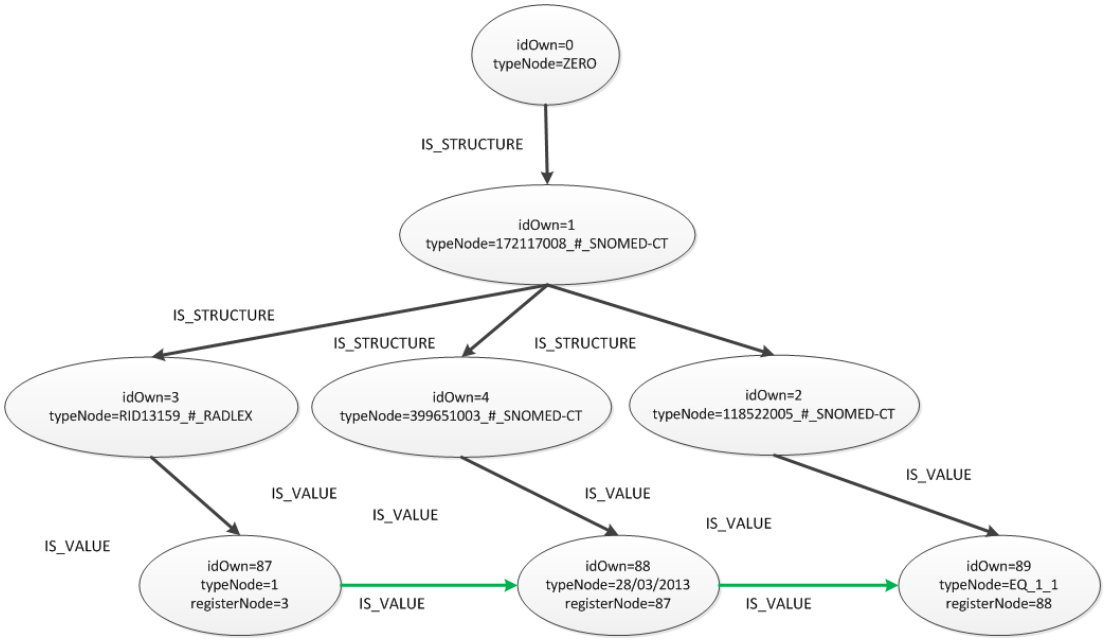

Figura 60 – Representación de la cabecera de una instancia en la ontología correspondiente.

En un principio podemos realizar la consulta de la siguiente forma:

```
start n1=node(*)
match n1-[r1]->n2
where n1.typeNode='ZERO'
and n2.typeNode='172117008_#_SNOMED-CT'
with n2
match n2-[r2]->n3
where n3.typeNode='RID13159_#_RADLEX'
with n3
match n3-[r3]->n4
where n4.typeNode='1'
with n4
match n4-[r4]->n5
where n5.typeNode>'20130101'
with n5
match n5-[r5]->n6
return n6.typeNode as Identificador_Informes;
```

La consulta funciona de la siguiente forma:

```
start n1=node(*)
```

La consulta comenzará por vincular un nodo (en este caso n1) a un punto de partido.

```
match n1-[r1]->n2
```

Tras ello dicho punto de partida tendrá una relación (una arista denominada r1) con otro nodo (o varios, que están representados por n2).

```
where n1.typeNode='ZERO'
and n2.typeNode='172117008_#_SNOMED-CT'
```

Y el nodo inicial (n1) será el nodo **"ZERO"**, que es el nodo inicial en Neo4j, y el siguiente nodo (n2) será el que identifica a la exploración de la mama, de esta forma fijaremos que la ontología del grafo es del tipo exploración mamaria.

```
with n2
```

Los nodos resultados obtenidos (n2), que en este caso el único nodo devuelto será el `'172117008_#_SNOMED-CT'`, se comenzará una nueva consulta, usando estos como puntos de partida.

```
match n2-[r2]->n3
where n3.typeNode=’RID13159_#_RADLEX’
```

El nodo inicial (n2) tendrá una relación (r2) con otro nodo (o varios, representado en n3), siendo estos nodos finales los identificadores de los pacientes (solamente habrá un nodo así en esta posición de la estructura).

```
with n3
match n3-[r3]->n4
where n4.typeNode='1'
```

El nodo inicial será el nodo identificador de los pacientes (n3), o sea el nodo con valor `’RID13159_#_RADLEX’`, que tendrá una relación (r3) con una serie de nodos que son los nodos que identifican a los distintos pacientes (n4). En este caso buscamos el paciente con identificador `'1'`.

```
with n4
match n4-[r4]->n5
where n5.typeNode>'20130101'
```

Por la estructura que tiene la cabecera de los informes podemos, desde el identificador de un paciente (n4), acceder a las distintas fechas de los informes (n5), y a través de ellos a los identificadores de los informes. Con esto relacionamos (r4) los nodos identificadores de los pacientes encontrados (n4) con las fechas de las exploraciones (n5) realizadas a partir del 01/01/2013.

```
with n5
match n5-[r5]->n6
return n6.typeNode as Identificador_Informes;
```

Con los nodos de la fechas de las exploraciones obtenidos (n5) podemos obtener los identificadores de los informes (n6) a través de la relación (r5) que tienen con ellos. Lo siguiente es devolverlos como resultado.

Esta sería la forma más sencilla de realizar la consulta, pero no la más óptima por las siguientes cuestiones:

- Al implementar una ontología en cada recurso, no es necesario buscarla como punto de entrada, ya que la consulta será realizada en ella misma.
- Al realizar comparaciones del tipo `n3.typeNode = 'RID13159_#_RADLEX'` estamos filtrando un conjunto de nodos que puede ser enorme, por lo que este proceso puede ser lento.
- Cada vez que se ejecuta una consulta ha de utilizar elementos de la estructura de la ontología, por lo que en cada consulta ha de buscar siempre los mismos elementos.
- En esta consulta se han ido buscando los nodos a través de relaciones simples, o sea dando saltos con 1 arista de profundidad.

Con estos problemas detectados, y con la solución propuesta podemos redefinir la consulta de la siguiente forma:

```
match n1-[r1:IS_VALUE]->n2:VALUE-[r2:IS_VALUE]->n3-[r3:IS_VALUE]->n4
using index n2:VALUE(typeNode)
where ID(n1)=5 and
n2.typeNode='1' and
n3.typeNode>'20130101'
return n4;
```

En esta solución propuesta no hemos usado como punto de entrada el identificador de la ontología, ya que el recurso en sí solo mantiene ontologías de un tipo en concreto.

```
where ID(n1)=5 and
```

Además de eso, gracias al servicio de cache interno, podemos obtener el número que identifica internamente al nodo `'RID13159_#_RADLEX'` (que es el nodo identificador del paciente) en Neo4j, de forma que es posible comenzar la consulta en dicho nodo directamente (que tiene el identificador `'5'`).

```
match n1-[r1:IS_VALUE]->n2:VALUE-[r2:IS_VALUE]->n3-[r3:IS_VALUE]->n4
```

A partir del nodo inicial podemos relacionarnos con los distintos nodos de forma óptima, manteniendo esto hasta una profundidad de 100 aristas. Como hemos asignado un tipo de relación a la arista, al ponerlo directamente en la relación, filtra otras aristas que tenga el nodo.

```
using index n2:VALUE(typeNode)
```

Podemos hacer uso de los índices creados sobre los *labels* (en este caso sobre *VALUE*), de forma que la búsqueda de los valores será mucho más rápida. Hay que destacar que, en este caso, solo es posible usar como criterio la igualdad, ya que con las fechas no hemos podido utilizar criterios de mayor o menor porque no lo permite el índice.

```
where ID(n1)=5 and
n2.typeNode='1' and
n3.typeNode>'20130101'
return n4 as Identificador_Informes;
```

El nodo n2 será filtrado usando el índice, mientras que el nodo n3 filtrará directamente cada valor, aunque no será lento, ya que la estructura propuesta solo permite que cada n2 tenga una relación con un n3, por lo que será rápida la comparación.

4.2.2.2. Descripción de los servicios.

Tal y como fueron definidos los servicios en [32] se adaptan perfectamente al modelo propuesto esta vez. No es necesario cambiar la funcionalidad de los mismos, simplemente adaptarlos a que existe una nueva implementación de *Indexer*.

Servicio DICOM Storage.

La finalidad del servicio es gestionar las instancias de los DICOM-SR en cualquiera de los repositorios, ya sea AMGA o Neo4j. Al encontrarse cada ontología en un recurso propio, se necesita una implementación basada en múltiples recursos, que es la actual. Se adapta perfectamente a la solución propuesta, por lo que no es necesario realizar ningún cambio.

En un futuro, al estar las ontologías separadas en distintos recursos y no tener relación entre ellas, sí se necesitará realizar los cambios oportunos en dicho servicio para adaptarlo a una solución que permita que las ontologías se relacionen entre ellas.

Servicio Storage Broker.

El servicio *Storage Broker*, implementado actualmente para AMGA, crea recursos que actúan como mediadores para cada tipo de ontología, permitiendo consultas simultáneas en todos los repositorios AMGA del sistema. Se adapta perfectamente a la solución propuesta, por lo que no es necesario realizar ningún cambio.

Servicio Ontologies Server.

El servicio *Ontologies Server* se encarga de almacenar las ontologías de los informes estructurados, así como de mantener actualizado el servicio del *DICOM Storage* y el *Storage Broker*. Se adapta perfectamente a la solución propuesta, por lo que no es necesario realizar ningún cambio actualmente.

4.2.3. Pruebas realizadas sobre el modelo 2.0.

4.2.3.1. Pruebas similares al modelo 1.0.

Se ha realizado el mismo conjunto de pruebas que en el modelo anterior para verificar una comparación entre la nueva propuesta, el modelo 1.0 y AMGA.

Tras las operaciones de inserción, los resultados obtenidos han mejorado un poco (apenas apreciable) debido a que ya no se construye en memoria para luego pasarlo a la base de datos, sino que se crean las instancias directamente en el grafo dentro de una transacción. Además de esto, al no almacenar todos los datos del fichero XML (solo se almacenan los valores de los distintos elementos) se crean menos nodos y aristas en el sistema.

En cuanto a las consultas propuestas en el modelo 1.0 se pueden reescribir de la siguiente forma para el nuevo modelo:

- Consulta 1: obtener el número de todas las exploraciones mamarias realizadas.

```
match n1-[r1:IS_VALUE]->n2
where ID(n1)=4
return count(n2) as total;
```

- Consulta 2: obtener el identificador de todos los informes que se le han realizado al paciente 461.

```
match n1-[r1:IS_VALUE]->n2:VALUE-[r2:IS_VALUE]->n3-[r3:IS_VALUE]->n4
using index n2:VALUE(typeNode)
where ID(n1)=5 and n2.typeNode='461'
return distinct(n4.typeNode) as Identificador_Informe;
```

- Consulta 3: obtener el identificador de todos los informes que se le han realizado al paciente 461 con fecha posterior al 01/01/2013.

```
match n1-[r1:IS_VALUE]->n2:VALUE-[r2:IS_VALUE]->n3-[r3:IS_VALUE]->n4
using index n2:VALUE(typeNode)
where ID(n1)=5 and
n2.typeNode='461' and
n3.typeNode>'20130101'
return distinct(n4.typeNode) as Identificador_Informe;
```

- Consulta 4: obtener los identificadores de todos los informes que se le han realizado al paciente 461, con fecha posterior al 01/01/2013 y que tenga en la mama derecha lesiones de tipo “masa” en la localización “retroareolar”.

```
match n1-[r1:IS_VALUE]->n2:VALUE-[r2:IS_VALUE]->n3-[r3:IS_VALUE]->n4
using index n2:VALUE(typeNode)
where ID(n1)=5 and n2.typeNode='461' and n3.typeNode>'20130101'
```

```
with n4
match n5-[r4:IS_VALUE]->n6:VALUE
using index n6:VALUE(registerNode)
where ID(n5)=8
AND n6.registerNode=n4.idOwn
with n4,n6
match n7-[r5:IS_VALUE]->n8:VALUE
using index n8:VALUE(registerNode)
where ID(n7)=9
AND n8.registerNode=n6.idOwn
AND n8.typeNode='1'
return distinct(n4.typeNode) as Identificador_Informe;
```

- Consulta 5: Obtener la fecha y el número de exploraciones mamarias que se le han realizado al paciente 461.

```
match n1-[r1:IS_VALUE]->n2:VALUE-[r2:IS_VALUE]->n3-[r3:IS_VALUE]->n4
using index n2:VALUE(typeNode)
where ID(n1)=5 and
n2.typeNode='461'
return n3.typeNode as fecha, count(n4) as total;
```

Los resultados obtenidos de las ejecuciones de estas consultas son los siguientes:

Consulta	Tiempo de búsqueda en Neo4j modelo 2	Tiempo de búsqueda en Neo4j modelo 1	Tiempo de búsqueda en AMGA
1	1,000 s	0,91 s	0,055 s
2	1,017 s	0,93 s	0,060 s
3	0,999 s	1,02 s	0,064 s
4	2,054 s	4,31 s	2,344 s
5	0,783 s	1,045 s	---

El resultado obtenido con las consultas ha mejorado considerablemente, obteniendo un modelo que proporciona una gran potencia y flexibilidad a las consultas. Por el contrario, se pierde la reutilización de los nodos, que no es necesaria en este modelo. La flexibilidad no varía respecto al modelo 1.0.

4.2.3.2. Pruebas de carga de datos al modelo 2.0.

Se ha realizado una prueba de carga masiva de datos sobre el modelo actual. Para ello se han insertado 500.000 registros de exploraciones mamarias, de 10.000 pacientes, comprendidos entre los años 2013 y 2015 (los datos son ficticios).

Esta carga ha supuesto en el grafo más de 68.000.000 de nodos, 69.000.000 de aristas y 141.000.000 de propiedades creadas, con un tamaño de 16,5 GB.

Sobre dicho grafo se ha realizado el conjunto de pruebas anteriormente citado. Con ello se ha podido verificar que los tiempos en la operación de inserción no varían prácticamente, mientras que en las consultas hemos obtenido los siguientes resultados:

Consulta	Tiempo de búsqueda en Neo4j modelo 2 con 500.000 registros	Tiempo de búsqueda en Neo4j modelo 2 con 9.000 registros	Tiempo de búsqueda en Neo4j modelo 1 con 9.000 registros	Tiempo de búsqueda en AMGA con 9.000 registros
1	2,342 s	1,000 s	0,91 s	0,055 s
2	0,654 s	1,017 s	0,93 s	0,060 s
3	0,927 s	0,999 s	1,02 s	0,064 s
4	3,921 s	2,054 s	4,31 s	2,344 s
5	0,879 s	0,783 s	1,045 s	---

Es necesario tener en cuenta que se trata de escenarios distintos, ya que, mientras las pruebas con 9.000 registros se han realizado en un entorno de máquinas con sistema operativo Scientific Linux 5.8 (Boron), con una capacidad de 4 GB de memoria RAM a 1066 Mhz, un procesador Intel Intel Xeon E5520 a 2.26 Ghz y disco duro de 10 GB a 10.000 r.p.m, las realizadas con 500.000 registros se han realizado en una máquina local, con sistema operativo Ubuntu 12.04 x64, con una capacidad de 3,5 GB de memoria RAM a 800 Mhz, un procesador Intel Core 2 Quad Q8200 a 2.33 Ghz y un disco de 1 TB a 5.400 r.p.m.

4.2.4. Conclusiones del modelo 2.0.

El modelo presentado en esta segunda aproximación cumple con las expectativas esperadas. El proceso de inserción es similar (e incluso algo más rápido) que en el modelo 1, por lo que ya de por sí es más rápido que AMGA.

Los resultados obtenidos con las consultas han mejorado considerablemente, principalmente por el nuevo sistema de búsqueda basado en el uso del servicio de cache interno, de forma que proporciona los identificadores de los nodos estructurales. Por contra, se pierde la reutilización de los nodos, pero la flexibilidad que proporciona el modelo para ser modificado no cambia.

En cuanto a las pruebas de carga realizadas se ha podido observar un leve aumento del tiempo de respuesta en algunas consultas, pero esto se debe principalmente a que las características de procesador y disco duro de la máquina local donde se ha realizado son menores que la máquina (o conjunto de máquinas en este caso) que se encuentran en el GRyCAP.

La implementación de cada nueva ontología en un grafo distinto permite adaptarse a la propuesta de múltiples recursos que funciona actualmente en el sistema, integrándose a la perfección con el modelo actual de TRENCADIS.

5. Conclusiones y aportaciones.

En esta tesis de master se ha realizado un estudio del estado del arte de las tecnologías actuales, que han permitido ampliar el ámbito de aplicación de TRENCADIS a un contexto Big Data, de forma que los informes estructurados DICOM-SR son almacenados en un modelo basado en grafos.

Se han propuesto dos modelos para verificar la viabilidad de la modificación respecto al modelo existente en TRENCADIS basado en AMGA, permitiendo prescindir de dicho *Indexer* durante el despliegue, y garantizando una completa compatibilidad con el servicio *DICOM Storage* actual, bajo un nuevo modelo de grafos dentro de un contexto de Big Data.

El modelo definido como versión 1.0 permite representar los ficheros XML de forma completa dentro de la base de datos Neo4j bajo un modelo de grafos, usando una jerarquía de tres niveles para representar la semántica, tanto de las ontologías como de las instancias. Esto nos lleva a tener una gran flexibilidad a la hora de poder modificar las ontologías, ya que podemos añadir o eliminar elementos de las mismas, de una forma muy sencilla. Este modelo también permite la reutilización de los nodos, de forma que las distintas ontologías y sus instancias reutilizan los nodos existentes y se encuentran relacionadas entre ellas.

El modelo 1.0 no cumple las expectativas esperadas, debido principalmente a que la reutilización de los nodos produce una sobrecarga de aristas en los mismos, afectando de forma negativa a las consultas, ya que, a cuanto mayor nivel de profundidad en el grafo, mayor latencia en las consultas. Dicha latencia también viene penalizada por el gran número de nodos existentes en las diferentes estructuras de las ontologías.

El modelo definido como versión 2.0, almacena los nodos que identifican a cada elemento que corresponde exclusivamente al informe DICOM-SR dentro del fichero XML, excluyendo el resto de elementos incluidos, y en el caso de las instancias, los valores propios de cada informe. Esto permite almacenar, tanto las ontologías como las instancias en un menor número de nodos y aristas, y sin hacer uso de la reutilización de los nodos, de forma que en cada nueva estructura generada en el grafo se crearán todos sus elementos.

El modelo 2.0 mantiene la flexibilidad del modelo 1.0, característica fundamental para poder añadir, modificar o eliminar elementos, tanto las instancias como las ontologías, de una manera muy sencilla. En contrapartida sí se perderá el principio de reutilización utilizado en el modelo 1, ya que no interesa por la latencia que añade a las consultas.

El modelo 2.0 obtiene tiempos razonables en las consultas, consiguiendo mejores resultados que el modelo actual implementado en AMGA, lo que hace que su integración en el servicio *DICOM Storage* esté justificada, permitiendo de esta forma

que los metadatos de los informes estructurados sean almacenados en un modelo de grafos bajo la base de datos Neo4j.

La solución propuesta se integra perfectamente con el conjunto de servicios implementados en TRENCADIS, ya que no se modifica ninguna funcionalidad de los servicios actuales, sino que será añadido un nuevo *Indexer* al componente *TRENCADIS Java API Indexer*.

Con dicha integración realizada se puede hablar de que TRENCADIS comienza una adaptación a los tiempos de Big Data, donde el tamaño de los datos, la desestructuración de los mismos en su conjunto, y su tratamiento y visualización jugarán un papel importante.

6. Artículos asociados.

El desarrollo de este proyecto ha motivado la creación de la siguiente publicación:

- Un artículo en el congreso *Ibergrid 2013* titulado "Graph Database for Structured Radiology Reporting", cuyos autores son Lorenzo Díaz, Damián Segrelles, Erik Torres y Ignacio Blanquer. ISBN: 978-84-9048-110-3. Página 61-73.

7. Trabajos futuros.

El siguiente paso a realizar en la implementación actual será el desarrollo de una solución que permita conectar todas las ontologías existentes. Esta permitiría la realización de consultas "transversales", de forma que será posible preguntar por datos de distintas ontologías al mismo tiempo.

Esta podría pasar por la generación de una meta-estructura que relacionara todas las ontologías, pasando a funcionar el servicio *DICOM Storage* como un servicio Grid *singleton* de nuevo, situándose todas las ontologías bajo el mismo grafo.

Otra solución podría pasar por crear un servicio Grid que realizara las consultas pertinentes a cada grafo y fusionara los resultados, según los criterios existentes de relación entre las distintas ontologías. Esto permitiría mantener el servicio *DICOM Storage* en su estado actual.

Otra de las mejoras futuras debería pasar por el tratamiento y visualización de los datos, o sea implementar una serie de procesos para poder realizar minería de datos dentro del grafo, ya que se trata de un campo imprescindible dentro de Big Data.

Bibliografía.

[1] Medical Imaging & Technology Alliance, "DICOM – Digital Imaging and Communications in Medicine", http://medical.nema.org

[2] D. A. Clunie - DICOM Structured Reporting. PixelMed Publishing. Bangor, Pennsylvania. 2000.

[3] Alfieri R, Cecchini R, Ciaschini V, dell'Agnello L, Frohner Á, L@rentey K, et al. From gridmap-file to VOMS: managing authorization in a grid environment. Future Gener Comput Syst (FGCS) 2005;21(4):549–58.

[4] Radiological Society of North America – "RadLex", https://www.rsna.org/RadLex.aspx

[5] World Health Organization – "International Classification of Diseases (ICD)", http://www.who.int/classifications/icd/en/

[6] International Health Terminology Standards Development Organisation – "SNOMED CT", http://www.ihtsdo.org/snomed-ct/

[7] Asociación Profesional del Cuerpo Superior de Sistemas y Tecnologías de la Información de la Administración del Estado – Big Data. Boletic nº 65 Abril 2013, pag 26-31.

[8] Stonebraker M - SQL databases v. NoSQL databases. Magazine Communications of the ACM. Volume 53 Issue 4, April 2010. Pages 10-11. ISSN:0001-0782 EISSN:1557-7317

[9] Blanquer I, Hernandez V, Segrelles D. - TRENCADIS. A grid architecture for creating virtual repositories of DICOM objects in an OGSA-based ontological framework, vol. 4345; 2006. p. 183–94. ISSN: 0302-9743, ISBN-10: 3-540- 68063-2, ISBN-13: 978-3-540-68063-5.

[10] CERN. – "Arda Metadata Catalogue Project", http://amga.web.cern.ch/amga/pages.html

[11] Koblitz B, Santos N, Pose V. - The AMGA metadata service. JGC 2007;6(1):61–76. ISSN: 1570-7873, 1572-9814.

[12] "gLite. LightWeight Middleware for Grid Computing". http://glite.web.cern.ch/glite/

[13] Neo4j. http://www.neo4j.org/

[14] Globus 4 toolkit. http://www.globus.org/toolkit

[15] Open grid services architecture (OGSA). http://www.globus.org/ogsa

[16] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, D. Snelling, Vanderbilt - Open Grid Services Infrastructure (OGSI). http://www.ggf.org/ogsi-wg

[17] "Web Services Description Language (WSDL)". http://www.w3.org/TR/wsdl

[18] Organization for the Advancement of Structured Information Standards. http://www.oasis-open.org.

[19] F. Magoulès, T.-M.-H.N., Lei Yu, Grid Resource Management - Toward Virtual and Services Compliant Grid Computing. Taylor & Francis Group, LLC, 2009.

[20] "LCG Middleware". http://lcg.web.cern.ch/LCG

[21] "Web Services Description Language (WSDL)". http://www.w3.org/TR/wsdl

[22] D. Box, D.Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H.Nielsen, S. Thatte, D. Winer, Simple Object Access protocol (SOAP) 1.1, Note 20000508, 2000

[23] B.Sotomayor, L.Childers – Globus Toolkit 4: Programming Java Services. Morgan Kaufmann Publishers, 2006.

[24] B. Koblitz, S. Ahn, A. J. Boloori, T. Calanducci, C. Cherubino, S. Hwang, N. Kim, S. Scifo - The gLite AMGA Metadata Catalogue. EGEE Conference, Istanbul, September 2008.

[25] Domínguez M. - Introducción al Middleware gLite 3.1. 1er Tutorial Nacional de Tecnología GRID. 16 – 20 Nov. 2009, CUBAENERGIA. C. Habana, Cuba.

[26] A. Delgado Peris, P. Méndez Lorenzo, F. Donno, A. Sciabà, S. Campana, R. Santinelli - LCG-2 User Guide. Document identifier: CERN-LCG-GDEIS-454439, August 2005.

[27] B. Revert. DICOM Cookbook. Philips Medical System Nederland, 1997.

[28] H. K. Huang. "PACS and Imaging Informatics: Basic Principles and Applications". ISBN: 0471251232. Editorial Wiley-Liss

[29] R. Noumeir - Benefits of the DICOM Structured Report. Journal of Digital Imaging. December 2006, Volume 19, Issue 4, pp 295-306.

[30] A. Lith, J. Mattsson - Investigating storage solutions for large data. Department of Computer Science and Engineering. Chalmers University Of Technology. Göteborg, Sweden, 2010

[31] D. Segrelles, M. Caballer, E. Torres, G. Moltó, I. Blanquer - Platform for Easing Deployment and Improving the Availability of TRENCADIS Infrastructure. Institute of Instrumentation for Molecular Imaging (I3M), Universitat Politècnica de València (UPV). Valencia, Spain, 2013.

[32] J. Salavert, D. Segrelles, I. Blanquer – Desarrollo e integración en TRENCADIS de servicios GRID para el almacenamiento, indexación y búsqueda de objetos DICOM-SR asociados a estudios de imágenes DICOM de distintos centros hospitalarios empleando componentes de gLite. Institute of Instrumentation for Molecular Imaging (I3M), Universitat Politècnica de València (UPV). Valencia, Spain, Noviembre 2011.

[33] "Camino en teoría de grafos" - http://es.wikipedia.org/wiki/Camino_%28teor%C3%ADa_de_grafos%29

[34] Rada Hussein, MSc; Uwe Engelmann, PhD; Andre Schroeter, MSc Hans-Peter Meinzer, PhD. - DICOM Structured Reporting. Division of Medical and Biological Informatics, H0100, German Cancer Research Center, Im Neuenheimer Feld 280, D-69120 Heidel- berg, Germany. RSNA, 2004