UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

etsinf

DIPLOMA DE ESPECIALIZACIÓN EN BIG DATA

TRABAJO FIN DE DIPLOMA

GENERACIÓN DE INTELIGENCIA RELACIONAL CON CYPHER

Curso 2014-2015
1ª Edición

autoritas
nuevas ideas, nuevas soluciones

AUTOR
Víctor Núñez Monsálvez

DIRECTOR
Francisco Manuel Rangel Pardo

Agradecimientos

A Kico Rangel, mi tutor, por guiarme en este reto

A Andrés Ramos, mi compañero de diploma, por su ayuda técnica

A mis padres por su infinita paciencia y comprensión

TABLA DE CONTENIDOS

Indice de tablas

Indice de figuras

1.Introducción

El marco actual de gestión de datos demanda un tipo de base de datos que se ajuste a los nuevos patrones de generación de datos que han venido proliferando en años recientes. Dichos patrones están muy influenciados por la madurez de la web 2.0 y la aparición y uso extensivo de nuevos dispositivos móviles con conexión a Internet tales como smartphones, tablets, smartwatches, relojes deportivos GPS o pulseras que monitorizan nuestras constantes vitales.

Por un lado, dichos dispositivos constantemente vuelcan datos o generan datos en modo de posts en blogs o comentarios en redes sociales por poner un ejemplo. Esto supone compartir datos personales y opiniones bien con otros usuarios directamente o través de aplicaciones.

Por otro lado, la web 2.0 supone una evolución en la madurez de Internet que se caracterizan porque los usuarios pasan a jugar un papel activo en la generación de contenidos hacia la red y hacia otros usuarios. Este término, popularizado por Tim O'Reilly y Dale Dougherty, se apoya en la constante evolución tecnológica que se ha venido produciendo desde mediados del siglo XX. Esta evolución ha facilitado el acceso del usuario final a la red y que sea sencillo generar contenidos en cualquier momento y desde cualquier sitio. Por lo tanto, se ha acercado la red al usuario final de tal manera que adquiere un papel activo en la generación de contenidos.

Esta revolución recuerda a la invención del ordenador personal que acercó el ordenador al usuario cuando hasta entonces solo había grandes ordenadores que podían ocupar un planta entera de un edificio y se dedicaban solo a temas científicos o de seguridad. Así pues, el término web 2.0 enfatiza el cambio de la relación del usuario con la red y se caracteriza por el contenido generado por el usuario, la usabilidad y la interoperabilidad [1]. Sin embargo, no hay que cometer el error de olvidarse de toda la tecnología que lo ha hecho y lo hace posible aunque ahora pueda parecer más transparente para el usuario común de la red.

Debido a estas razones, se ha añadido una nueva componente en los datos susceptible de ser analizada para fines tan dispares como de negocio o de seguridad. Analizar la interconexión, implícita o explícita, entre los datos es la nueva necesidad que ha aparecido como consecuencia del cambio de patrón de conducta y de relación entre individuos favorecido por las nuevas tecnologías y que ha cambiado el enfoque del usuario hacia la red.

En resumen, un rasgo común de la mayoría de datos generados hoy en dia en Internet es la gran interconexión entre ellos. Así pues, los modelos de datos de las bases de datos que los albergan también han tenido que adaptarse a esta tendencia así como las herramientas que los explotan para extraer conocimiento.

Las antiguas bases de datos relacionales se muestran ineficientes y suponen un gran coste computacional para manejar datos cuya principal característica es cómo se relacionan entre

sí y no solamente la información que contienen. El énfasis a la hora de tratar estos datos para extraer conocimiento debe ser puesto ahora en la relación y para tal propósito las bases de datos de grafos son las más indicadas [2].

Además, las bases de datos de grafo deben ser complementadas con una adecuada representación gráfica en grafo de la información que contienen para hacerlas útiles. Primero, se interrogará a la base de datos mediante una consulta en forma de patrón de búsqueda sobre lo que se busca. Después, la información recuperada debe ser visualizada de manera que sea más comprensible y ayude al analista a extraer conocimiento.

A continuación, se describen los motivos por los cuales se ha decidido llevar a cabo este proyecto y el marco en el cual se engloba.

1.1. Motivación

El presente trabajo se engloba dentro de un proyecto de evolución de la herramienta Cosmos Intelligence de la empresa Autoritas Consulting. Esta empresa se dedica a la consultoría estratégica y global y su negocio se centra en la generación de inteligencia para las organizaciones [3].

Uno de los aspectos a potenciar hoy en día es el de integrar la inteligencia social en los procesos de negocio para obtener ventaja competitiva. En otras palabras, integrar los nuevos usos de la red y los datos relacionados, abundantes y públicos como un aspecto más de la estrategia global de las empresas de una manera sistemática y eficiente.

La inteligencia social se define como la capacidad de navegar efectivamente y negociar entornos y relaciones sociales complejas [4]. Originalmente fue definida por el psicólogo Edward Thorndike en 1920 como “la habilidad de entender y manejar a hombres y mujeres, chicos y chicas, para actuar sabiamente en las relaciones humanas”.

La idea que subyace es que si entendemos mejor al ser humano y cómo se relaciona, también se podrá entender mejor sus necesidades en cada momento o, incluso, adelantarse a ellas (enfoque pro-activo).

Por ejemplo, un uso de la inteligencia social podría ser el de analizar de qué está hablando la gente en Internet para poder identificar necesidades en el mercado o averiguar el grado de aceptación de las marcas y los productos. Es un enfoque pro-activo que puede contribuir a anticiparse al mercado, lo cual ayuda a la empresa en su estrategia comercial y a minimizar riesgos.

Actualmente, la herramienta Cosmos Intelligence recupera datos de diferentes fuentes, los almacena y los indexa mediante índices Lucene para analizarlos numéricamente y gráficamente. Apache Lucene es una librería de motor de búsqueda de texto de alto rendimiento escrita en Java [5]. La diversas fuentes de información empleadas comprenden,

entre otras, las redes sociales, los blogs, los periódicos y los foros. Se analiza la interacción de los datos desde diferentes vertientes.

Primero se selecciona el proyecto y después se selecciona las fuentes de información. Una vez están cargados los datos, se pasa a analizar la información desde una perspectiva textual y gráfica, mediante su visualización en modo de grafo.

Por ejemplo, para el caso de Twitter se tienen en cuenta aspectos como qué usuario sigue a quién (follower) o qué usuario retuitea a quién (retweet).

La herramienta que se propone en el presente trabajo se integra en la fase de análisis, dentro de la parte de comprensión del entorno (escucha y análisis), de la metodología seguida por la empresa Autoritas Consulting para lograr el posicionamiento deseado [3]. En la figura 1, se muestra todos los pasos de la metodología.

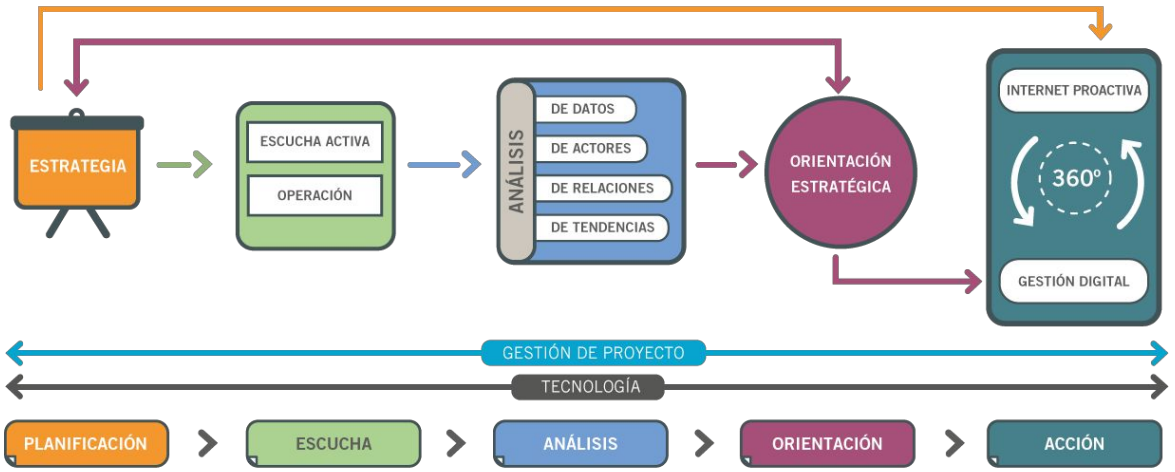

Figura 1. Ciclo de inteligencia de Autoritas Consulting.

El primer paso en la evolución, consistió en emplear bases de datos de grafos en vez de índices Lucene. Así pues, a partir de la ruta del índice Lucene, se genera una base de datos de grafos Neo4j en la ruta especificada empleando herramientas del API de Neo4j para embeberla en aplicaciones Java [6]. Todo este proceso está descrito extensamente en[7].

El segundo paso y objetivo de este trabajo, es el de explotar gráficamente dicha base de datos Neo4j una vez ya se ha generado y se dispone de una ruta a ella. El propósito es el ofrecer en un mismo portlet una consola donde el usuario escriba consultas Cypher para atacar a una base de datos de grafos en Neo4j y la visualización avanzada en un grafo de la información devuelta por el servidor Neo4j en la cual se puede interactuar con ella mediante el ratón.

En el siguiente apartado, se hará una introducción a las bases de datos de grafos, se realizará una comparativa y es presentará Neo4j, la opción elegida en este trabajo.

2. Bases de datos de grafos

2.1.Definición

Las bases de datos de grafos son un tipo de bases de datos que emplean estructuras de grafo y contra la cual se realizan búsquedas semánticas [1]. Por lo tanto, este tipo de base de datos está basadas en la teoría de grafos.

Sus componentes básicos son los nodos, las relaciones y las propiedades (en nodos y relaciones), a diferencia de las bases de datos relacionales clásicas, donde la información se estructura en tablas relacionadas y dichas tablas contienen registros (filas) con el mismo tipo de información (columnas o campos).

El elemento fundamental y diferenciador en las bases de datos de grafos es el énfasis en la relación más que ser meros contenedores de información independientes o poco relacionados entre sí. La relación forma parte del núcleo de la base de datos. La estructura de grafo interna favorece que se ajusten a este propósito. Esta característica hace que las bases de datos de grafos sean especialmente eficientes en el manejo de las relaciones mientras que para las bases de datos relacionales es un proceso muy costoso.

En las bases de datos de grafos, la información está contenida en entidades (nodos y relaciones) que conforman un grafo. Los nodos están relacionados entre sí por relaciones y, ambos, contienen cierta información en forma de propiedades.

Por un lado, las bases de datos relacionales contienen tablas con muchos registros con el mismo tipo de información (campos) y relacionadas entre sí mediante cierto campos claves entre tablas. Los registros de una misma tabla no están relacionados entre sí, sino que pertenecen a la misma tabla y contienen el mismo tipo de información.

Por otro lado, en las bases de datos de grafos la información está contenida en los nodos y las relaciones: mediante las propiedades de nodos y relaciones y la misma conexión entre nodos mediante relaciones. Los nodos pueden estar conectados con cualquier tipo de nodo para el cual exista un tipo de relación en el diseño de la base de datos. Así, existe la posibilidad de que un tipo de nodo se relacione con otro tipo de nodo mediante diferentes tipos de relaciones e incluso con nodos de su mismo tipo si existe algún tipo de relación que los permita.

A diferencia de las bases de datos relacionales, el hecho de que el diseño de la base de datos establezca una serie de relaciones entre tipos de nodos, no es obligatorio que existan dichas relaciones ni una limitación en número. En el caso relacional, cada registro de una tabla debería estar relacionado con otro u otros (según el tipo de relación) de otra tabla si el esquema de la base de datos así lo establece. Las relaciones entre nodos se van creando a medida que van apareciendo en la realidad para ajustarse a ella de una manera eficiente.

Como consecuencia, el manejo de las relaciones es muy eficiente. Por ejemplo, permite conocer, de manera muy eficiente, las relaciones entre nodos no directamente relacionados

sino a través de otros nodos y relaciones, incluso de otro tipo. Un ejemplo clásico que este tipo de base de datos podría abordar con mayor eficiencia sería el de averiguar "¿cuáles son los amigo de mis amigos?". Para que las bases de datos relacionales pudieran replicar el concepto de las bases datos de grafos deberían emplear extensivamente JOINS, lo cual ralentizaría su rendimiento.

Por todo lo anteriormente expuesto, las bases de datos de grafos son especialmente adecuadas para representar datos muy relacionados entre sí como, por ejemplo, los datos provenientes de redes sociales. Además, su estructura de grafo es más sencilla y directa de trasladar (mapear) a la estructura de aplicaciones orientadas a objetos, como aquellas escritas en Java. De hecho, la mayoría de estas bases de datos están escritas y/o tienen interfaces de cliente en lenguajes orientados a objetos como Java, JavaScript o C++.

Este tipo de bases de datos son encuadradas en el grupo de bases de datos NoSQL en contraposición a las bases de datos relacionales clásicas [8]. Las bases de datos NoSQL se clasifican en los siguientes grupos según el tipo de modelo de datos:

- *columna*
- *documento*
- *clave-valor*
- *de grafo*

Existe otro tipo, denominado multi-modelo, que son compatibles con varios de los modelos de datos no estructurados antes citados.

Un breve listado de bases de datos NoSQL según el modelo de datos implementado se muestra en la tabla 1.

Modelo de datos	Ejemplos de bases de datos NoSQL
Columna	Accumulo, Cassandra, Druid, Apache HBase, Vertica
Documento	Apache CouchDB, Clusterpoint, Couchbase, DocumentDB, HyperDex, Lotus Notes, MarkLogic. MongoDB, OrientDB, Qizk
Clave-Valor	Aerospike, CouchDB, Dynamo, FairCom c-treeACE, FoundationDB, HyperDex, MemcacheDB, MUMPS, Oracle NoSQL Database, OrientDB, Redis, Riak
Grafo	**Allegro, InfiniteGraph, MarkLogic, Neo4j, OrientDB, Virtuoso, Stardog**
Multi-modelo	ArangoDB, FoundationDB,MarkLogic, OrientDB

Tabla 1. Ejemplos de bases de datos NoSQL según modelo de datos

En negrita se han resaltado las bases de datos de grafos y, en azul, la opción escogida en este trabajo: Neo4j.

En la tabla 2 se muestra una comparativa realizada por Ben Scofield entre los distintos tipos de bases de datos NoSQL [9]. Para cada tipo de base de datos NoSQL se han valorado diversos aspectos a tener en cuenta a la hora de escoger un tipo u otro en función de las necesidades del problema a describir.

Modelo de Datos	Rendimiento	Escalabilidad	Flexibilidad	Complejidad	Funcionalidad
Clave-Valor	Alto	Alta	Alto	Ninguna	Variable (ninguna)
Orientada a Columna	Alto	Alta	Moderada	Baja	Mínima
Orientada a Documento	Alto	Variable (Alto)	Alto	Baja	Variable (baja)
Base de datos de grafo	**Variable**	**Variable**	**Alta**	**Alta**	**Teoría de Grafo**
Base de datos relacional	Variable	Variable	Baja	Moderada	Álgebra relacional

Tabla 2. Comparativa de las base de datos de grafo

Uno de los inconvenientes para los desarrolladores de bases de datos relacionales es el cambio radical de concepto a la hora de abordar las bases de datos de grafos ya que el concepto es absolutamente diferente tanto al diseñarlas como luego al realizar consultas.

Otro inconveniente es la escalabilidad. Tradicionalmente, las bases de datos de grafos escalaban en vertical pero no horizontalmente. Esto producía que buscar nodos en diferentes máquinas ralentizara fuertemente el proceso. Esta limitación desaconsejaba el uso de estas base de datos para tamaños mayores.

Sin embargo, algunos vendedores han abordado ya este problema. Por ejemplo, Neo4j escala dignamente tanto vertical como horizontalmente de modo que puede funcionar con 2GB o 200GB de RAM independientemente del tamaño del conjunto de datos [10].

2.2.Principales bases de datos de grafos

En este punto se realizará un breve repaso a las bases de datos de grafos. En primer lugar, en la tabla 3 se relacionan diversas bases de datos de este tipo con el lenguaje de consulta. Como ya se comentó, predominan los lenguajes de programación orientados a objetos y, en concreto, Java.

Nombre	Lenguaje de consulta (Query Language)
AllegroGraph	SPARQL
DEX/Sparksee	C++, Java, .NET, Python
FlockDB	Scala
IBM DB2	SPARQL
InfiniteGraph	Java
MarkLogic	Java, JavaScript, SPARQL, XQuery
Neo4j	Java
OWLIM	Java, SPARQL 1.1
Oracle	SPARQL 1.1
OrientDB	Java
Sqrrl Enterprise	Java
OpenLink Virtuoso	C++, C#, Java, SPARQL
Stardog	Java, SPARQL

Tabla 3. Bases de datos de grafo y su lenguaje de consulta

En segundo lugar, en figura 2 se muestra una evolución del “ranking” de popularidad de las bases de datos de grafos a fecha de diciembre 2015 en los últimos 3 años según el sitio web db-engines.com especializado en los sistemas de gestion de bases de datos relacionales y NoSQL [11].

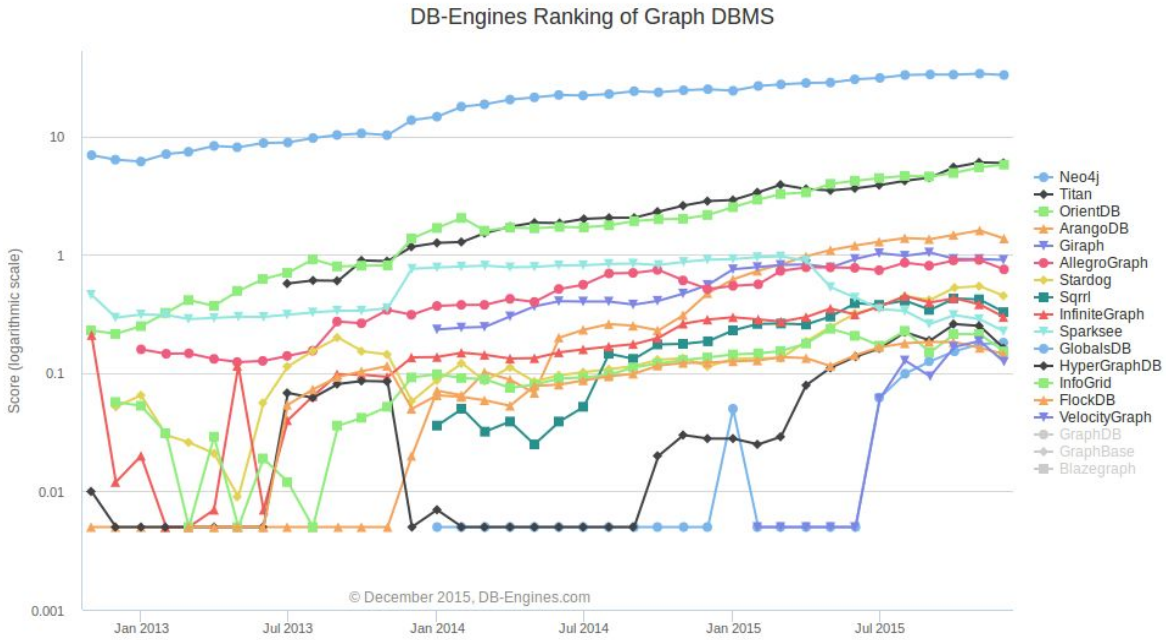

Figura 2. Evolución de la popularidad de las bases de datos de grafos.

Neo4j es la base de datos de grafo más empleada en el mundo con diferencia, sólo seguida de lejos por Titan y OrientDB, ambas desarrolladas con Java también. En el siguiente punto, se describirá Neo4j, la base de datos de grafos empleada.

Entre las razones para escoger Neo4j estuvieron que es la más empleada, con diferencia, en los últimos tiempos, que escala bien para grandes conjuntos de datos y que está escrita sobre Java. Esto último facilita su integración con las diversas tecnologías y lenguajes de programación empleadas en el proyecto como:

- *Java* para la parte de servidor, ya que se dispone de funciones para embeber una base de datos Neo4j en aplicaciones Java.

- La librería *vis.js* de Javascript para visualización y los componentes *PrimeFaces* (desarrollados en JSF) para el terminal en la parte de cliente.

2.3.Neo4j

Una vez introducidas las bases de datos de grafos y fundamentada la elección de Neo4j, en este apartado se darán algunos detalles más sobre ella.

En primer lugar, Neo4j está financiada por Neo4j Technology y se empezó a desarrollar en 2003, aunque no se dio a conocer públicamente hasta 2007 [12]. Hoy en día, Neo4j es la base de datos de grafos más popular y es empleada por miles de empresas y organizaciones de todo tipo, como Cisco, Walmart o Infojobs [13]. Algunos campos en los que se aplica son [14]:

- *Detección de fraude*: El análisis en tiempo real de datos relacionados destapa redes de fraude y otras estafas (scam) sofisticadas.
- *Búsqueda basada en grafo*: Librería de medios construida alrededor de las relaciones entre los activos digitales y sus atributos.
- *Gestión de acceso e identidad*: Quién eres, cómo perteneces y qué se te permite depende de las relaciones entre tú, una organización y el sistema.
- *Gestión de datos maestros* (master data): La organización y las líneas de producto son inherentemente representados como grafos: profundas jerarquías con conexiones arriba-abajo, laterales y diagonales.
- *Red y operaciones IT* (Information Technology): Las interconectadas capas física, virtual y de aplicación de una red son perfectamente modeladas en un exhaustivo grafo Neo4j.
- *Recomendaciones en tiempo real*: Conecta los puntos de aparentemente no relacionados intereses y relaciones para realizar recomendaciones que equilibran lo nuevo con lo familiar.
- *Redes sociales*: Familia, amigos y seguidores (followers) se extienden en un grafo social que revela patrones de comportamiento similar, influencia y grupos implícitos.

En segundo lugar, Neo4j es un motor de base de datos de grafo escrito sobre JVM (Java Virtual Machine) altamente escalable y de código abierto que soporta ACID (Atomicidad, Consistencia/Integridad, Aislamiento, Durabilidad/Persistencia), un conjunto de propiedades que garantiza que las transacciones en bases de datos son procesadas fiablemente [15]. A modo de curiosidad, las transacciones son denominadas recorrido (“traversal” en inglés) en el ámbito de las bases de datos de grafos.

Neo4j consta de dos ediciones, la Community (comunidad) y la Enterprise (empresa). La edición Community es una versión básica con la base de datos de grafo de alto rendimiento y totalmente transaccional-ACID.

La edición Enterprise, además de las funcionalidades de la edición Community provee otras adicionales como:

- *clustering escalable*
- *tolerancia ante fallos (fail-over)*
- *alta disponibilidad*
- *recuperación en vivo (live backup)*
- *monitorización completa*

Neo4j se instala como servidor sobre todo tipo de sistemas operativos. Por defecto, el servidor Neo4j viene con una interfaz interactiva web de base de datos conectada al puerto 7474 (http://localhost:7474). Ésta posibilita el lanzar consultas contra la base de datos de grafo y visualizar los resultados tanto gráficamente (vista 'Graph', grafo) como numéricamente (vista 'Rows', filas):

- En la *vista numérica*, se muestra la información ordenada en tablas exportables tal y como se indicó en la consulta en el apartado RETURN. Por ejemplo, se podría consultar la información de cada par de nodos conectados por una relación con los valores de sus respectivas propiedades (tanto para los nodos como para la relación).
- En la *vista gráfica*, se puede interactuar con el grafo dibujado a partir de la información recuperada de la consulta empleando el ratón. Por ejemplo, se podría "enfocar", mediante el ratón, un nodo o una relación y ver el valor de sus propiedades. También, se podría "pinchar" en un nodo o en una relación para seleccionarlo y modificar su posición.

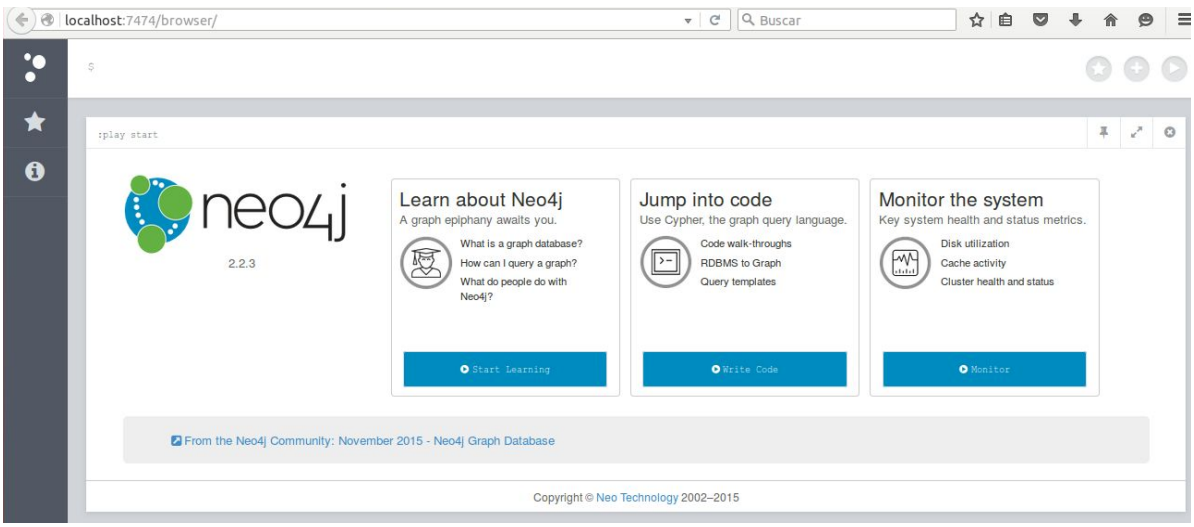

Figura 3. Vista inicial de la interfaz web de Neo4j.

En la figura 3 se muestra la vista inicial de la interfaz web del servidor Neo4j. En la parte superior se encuentra la línea de comando para escribir la consulta. Esta puede constar de varias líneas si fuera necesario.

Finalmente, el lenguaje de consulta sobre la base de datos de grafo Neo4j es Cypher [16]. En el siguiente punto se realizará introducción a Cypher.

2.4.Cypher

Este es un lenguaje declarativo inspirado en SQL exclusivo para Neo4j. Sin embargo, Neo4j y otros han desarrollado la iniciativa OpenCypher para extender el uso de Cypher a otras bases de datos de grafo [17].

La sintaxis de Cypher ofrece un modo familiar de emparejar patrones de nodos y relaciones en el grafo. En otras palabras, Cypher permite describir lo que queremos en vez de cómo hacerlo exactamente. Algunas operaciones que se pueden realizar mediante Cypher son seleccionar, insertar o actualizar entidades del grafo (nodos y relaciones).

Para representar a los nodos, se emplean los paréntesis, que asemejan los círculos de una representación gráfica de los nodos en un grafo. Para hacer referencia a los nodos, se puede agregar un identificador. Este, es aconsejable que tenga un nombre descriptivo del tipo de entidad que se describe.

Por ejemplo, si el dominio de la base de datos fuera Twitter, se podría emplear los identificadores “user” para los nodos que representaran a los usuarios mientras que “tweet” a los nodos que fueran los documentos (tuit) que generan en dicha red social.

Así pues, para encontrar los tuits relacionados con todos los usuarios se podría emplear el patrón (user)-->(tweet).

Para acceder a sus propiedades, habría que emplear el identificador, seguido de un punto y el nombre de la propiedad, por ejemplo “name” (nombre).

user.name

Para preguntar por los nombres de todos los usuarios habría que emplear la siguiente consulta

MATCH (user) RETURN user.name

El uso de mayúsculas o minúsculas es indiferente para las cláusulas de Cypher como MATCH o RETURN pero se ha elegido mayúsculas por claridad. En cuanto a los identificadores de nombres y relaciones y las propiedades, sí que Cypher es sensible a ello.

Para clarificar más aún lo que buscamos, se podría especificar el tipo de relación que se busca. Esto se realiza, escribiendo, entre corchetes, el nombre de la relación precedido de dos puntos (:).

Mediante la siguiente consulta se recupera la propiedad nombre de usuario (“username”) de los usuarios (nodos tipo “User”) relacionados con otros porque le han retuiteado (relación tipo “RT”):

MATCH (u1:User)-[:RT]->(u2:User) RETURN u1,u2

Se ha empleado las etiquetas u1 y u2, puesto que sin ellas se estarían indicando en el patrón aquellos usuarios relacionados consigo mismos por una relación de tipo RT.

A modo de resumen, en la figura 4, se puede ver una equivalencia entre una relación “LIKES” entre dos nodos, “a” y “b”, de un grafo con el modo mediante el cual se expresaría como patrón con la sintaxis Cypher.

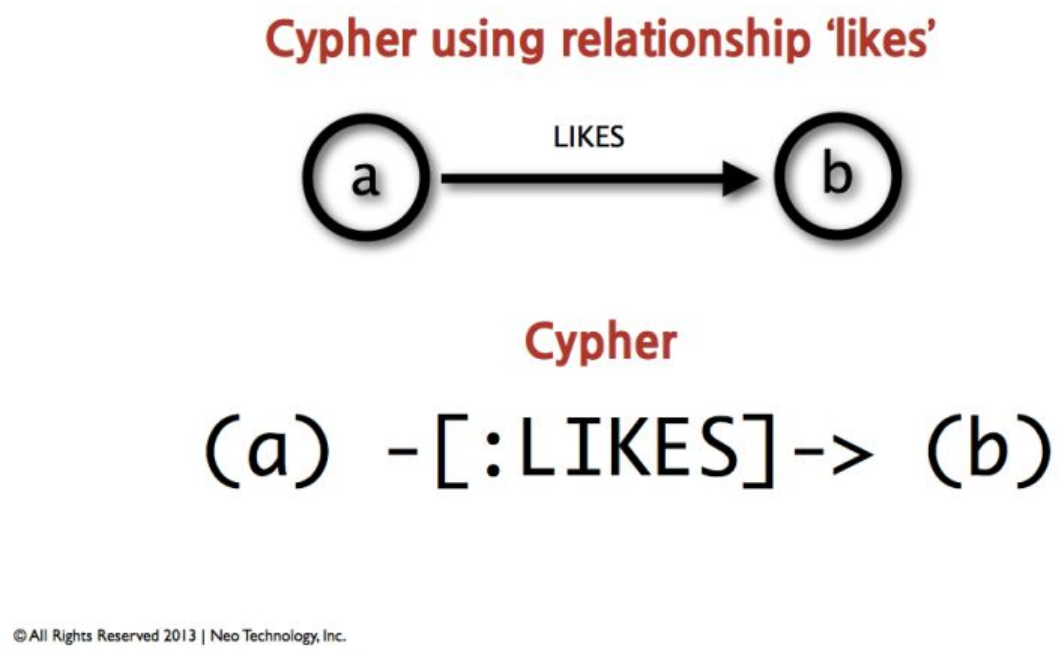

Figura 4. Ejemplo de equivalencia entre una relación y su patrón con Cypher.

A continuación se describe brevemente un consulta Cypher realizada en la aplicación web de Neo4j.

2.5. Ejemplo de consulta

En la figura 5, se muestra un ejemplo de consulta. En esta consulta se pregunta (match) por todos los nodos de tipo User relacionados entre sí de manera directa, cualesquiera que sea la relación. Además, se pide que devuelva (return) la información de los nodos y de las relaciones y se limita (limit) los resultados a diez relaciones.

```
$ match (n:User)-[r]->(m:User) return n,r,m limit 10
```

Figura 5. Ejemplo de consulta Cypher en la interfaz web de Neo4j.

Se interroga con dicha consulta una base de datos de grafo extraída de la herramientas Cosmos Intelligence de Autoritas Consulting. El dominio es la red social Twitter, los nodos serán de tipo “User” y las relaciones no se fijan en la consulta sino que se pregunta por cualquier relación entre dos nodos de tipo “User” (usuario de Twitter).

En la figura 6, se muestra la vista de Filas (Rows) del resultado de la consulta anterior. En concreto, se muestra la primera fila de los resultados, la cual está constituida por tres tablas: las correspondientes a los dos nodos y a la relación que los une. La tabla de la izquierda corresponde al nodo de tipo “User” con nombre de usuario “jabiriva” y la de la derecha al nodo de tipo “User” de nombre de usuario “mediterraneoo”, mientras que la del centro a la relación de tipo “RT” (retuit).

A parte de la propiedad “username”, los nodos de tipo User tienen las propiedades “lists”, “friends”, “followers” y “gender”. Las entidades relación, a parte del tipo, también pueden contener información adicional en forma de propiedad, lo cual enriquece más aún la expresividad del grafo. En el ejemplo, la relación solamente tiene como propiedad el tipo de relación (type).

Figura 6. Vista de Filas de la interfaz web de Neo4j.

En la figura 7, se muestra el grafo obtenida de la consulta. Se aprecia 10 nodos de tipo User (usuario) y 12 relaciones de tipo RT (retuit). El sentido de la relación RT expresa qué usuario ha retuiteado a qué otro usuario de la red social Twitter. A partir del grafo, se puede

deducir intuitivamente que ciertos usuarios concentran los retuits. De este hecho, se podría inferir que son “influencers”, es decir, personas cuya opinión marca tendencia.

Otro detalle a resaltar es cómo nodos del mismo tipo (User) están relacionados entre sí y esa relación podría ser de diferente tipo y contener información personalizada para cada una mediante las propiedades. En este punto, se puede vislumbrar en la práctica el gran “salto” conceptual que suponen las base de datos de grafo respecto de las bases de datos relacionales y su virtudes a la hora de representar gráficamente datos muy relacionados entre sí.

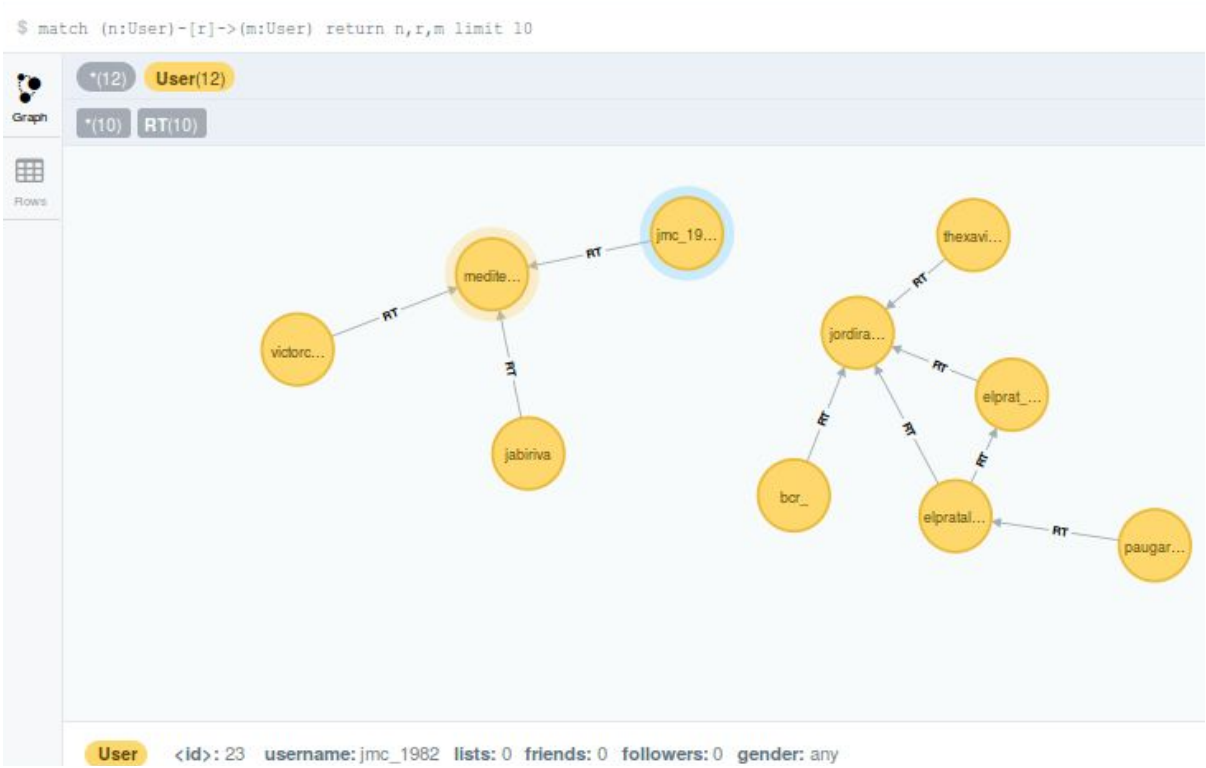

Figura 7. Vista de Grafo de la interfaz web de Neo4j.

Por último, al “enfocar” un nodo o una relación con el ratón, aparece un aura entorno a él y sus propiedades aparecen en la parte inferior del grafo. En el ejemplo, se “enfoca” el nodo de tipo usuario con nombre de usuario “jmc_1982” y aparece un aura azul en torno a él. Abajo del grafo, aparece el tipo de nodo (User), el id interno de la base de datos de grafos Neo4j (23) y las propiedades con sus valores: “username” (jmc_1982). “lists” (0), “friends” (0), “followers” (0), “gender” (any). Los nodos conectados con él, en este caso uno solo de tipo User y nombre de usuario “mediterraneoo”, aparecen resaltados con un aura roja.

3.Entorno de Trabajo

El propósito de este trabajo fue el de realizar un intérprete Cypher y presentar una visualización avanzada en forma de grafo de la información recuperada. Para ello, se integró en un mismo portlet una parte de consola, con el intérprete Cypher y otra parte de grafo. A continuación se aclaran algunos conceptos empleados en el trabajo:

- Un portlet es un componente software de interfaz de usuario conectable (modular) que es manejado y visualizado en un portal web o de empresa. Produce un fragmento de código de marcado (HTML, XHTML, WML) que es agregado al portal donde se incluye. Es un ejemplo de ingeniería software basada en componente. [18]

- El portal web es un tipo de sitio web que junta información de diferente fuentes de manera uniforme. Normalmente, cada fuente de información tiene su área dedicada en el portal para mostrar información (portlets). Es usual que el usuario pueda elegir qué fuentes quiere que se muestren. [19]

- El portal de empresa es un marco para integrar información, personas y procesos a lo largo de la organización de una manera similar al portal web pero acotado a los límites de dicha organización. [20]

Primero, se escogió Liferay como plataforma para construir el proyecto. En ella, se incluiría el portlet anteriormente comentado. Liferay Portal es un portal de empresa gratuito, de código abierto y escrito en Java. Gracias a su sistema de gestión de contenidos web integrados (built-in), posibilita la construcción de sitios web y portales mediante la agregación de temas, páginas, portlets y una navegación propia. [21]

Para Liferay existen dos ediciones:
- Community, base de la plataforma Liferay Portal.
- Enterprise, más orientada a entornos profesionales que requieren entornos estables y fiables.

En la tabla 4 se comparan ambas ediciones:

Community	Entreprise
Gratis	Bajo suscripción a Liferay Entreprise
Características soportadas por la comunidad	Releases estables con bugs solucionados
Para contribuir con los desarrollos de Liferay	Para desarrollar proyectos que pasan a producción
Instalaciones pequeñas y no críticas	Grandes instalaciones empresariales y con carácter crítico

Tabla 4. Comparativa entre las ediciones Community y Enterprise de Liferay Portal.

El primer paso para desarrollar el portlet fue instalar y configurar el entorno de desarrollo [22].

Para la edición Community de Liferay, se descargó Liferay 6.2 junto con Tomcat y Liferay Plugins SDK. Se descomprimió Liferay 6.2 y, en ese mismo directorio, se descomprimió Liferay Plugins SDK.

Para el desarrollo de la instancia de Liferay Portal se decidió usar Liferay IDE (Integrated Development Environment), una extensión para Eclipse IDE, puesto que soporta el desarrollo de proyectos de plugin para plataforma Liferay Portal. Se escogió la opción de instalar Eclipse junto con Liferay IDE (bundle). Una vez descargado, se descomprimió y se ejecutó el fichero ejecutable de Eclipse. Una vez en funcionamiento Eclipse, se seleccionó la perspectiva Liferay (Window->Open Perspective->Liferay) y se accedió a un entorno de desarrollo de Liferay desde Eclipse de una manera sencilla e integrada.

Cuando ya se instaló Eclipse y Liferay IDE conjuntamente, se tuvo aún que configurar Liferay IDE antes de poder desarrollar el portlet y probarlo. Los pasos fueron los siguientes:

- Liferay Plugins SDK: En Eclipse, se tuvo que indicar donde estaban instalados. (Preferences->Liferay->Installed Plugin SDKs, hacer clic en Add y en el diálogo Add SDK navegar hasta dar con la ruta donde fueron instalados, se acepta y se comprueba que se han añadido a la lista “Installed Plugin SDKs”).

- Configuración de Liferay Portal Runtime y del servidor: En este caso, se optó por el servidor de aplicaciones Tomcat, aunque no es la única opción posible. Se descargó Liferay junto con Tomcat como se comentó. Para esta configuración, hay que seguir en Eclipse la ruta “Window->Preferences->Server->Runtime Environments” y añadir un nuevo runtime de Liferay. Se seleccionó la ruta donde se instaló Liferay 6.2 y después la carpeta llamada “tomcat” más el número de versión. Si todo fue correcto, apareció un JRE (Java Runtime Environment), que automáticamente se seleccionó como el JRE lanzador del servidor.

En este punto, ya se estaba preparado para desarrollar y probar portlets para portales Liferay desde Eclipse IDE.

4.Solución Implementada

Una vez instalado y configurado el entorno de trabajo, se dividió el desarrollo en dos partes fundamentales:

- Cliente (archivo view.xhtml)
- Servidor (archivo terminalportlet.java)

El código tanto para la parte cliente como la parte servidor se encuentra en la parte de apéndices, al final de esta memoria.

Además, hay que introducir las librerías PrimeFaces y vis.js, que se emplearon en la parte cliente.

4.1.PrimeFaces

PrimeFaces es una librería de código abierto, escrita en Java, de componentes de interfaz de usuario para aplicaciones basadas en JSF (JavaServer Faces), una especificación Java para construir interfaces de usuario basadas en componentes para aplicaciones web. Fue creada por PrimeTek y actualmente ocupa el segundo lugar entre los “web framework” según la clasificación publicada por el sitio web devrates.com en base a la opinión de los propios usuarios [23].

Según sus creadores, PrimeFaces es una librería “ligera” con un único .jar y que no requiere configuración ni resolución de dependencias. Se puede descargar y añadir al classpath del proyecto [24]. En este caso, se eligió añadir una dependencia en el fichero ivy.xml del proyecto con el nombre de la librería, el nombre de la organización y la versión, para que el propio entorno de desarrollo Eclipse IDE se encargará de cargar la librería al “refrescar” el proyecto.

Los componentes PrimeFaces empleados en la parte cliente se tomaron de su escaparate [25] y fueron:

- *Panel* <*p:panel*>: Es un componente de agrupación con una cabecera, contenido y pie de página (footer).
- *Focus* <*p:focus*>: Gestiona el enfocado de elementos en formularios. Por defecto, el primer elemento de entrada visible y habilitado recibe el foco automáticamente. También es posible declarar explícitamente el componente que reciba el foco mediante el atributo “for”.

- *Terminal* <*p:terminal*>: Es una consola para introducir comandos que son manejados por un escuchador (listener) en el lado del servidor. El manejador de comandos se indica con el atributo “commandHandler”.

- *CommandButton* <p:commandButton>: Es un botón de comando que extiende el estándar h:commandButton (botón de envío) de HTML con ajax, procesamiento parcial y características para cambiar su apariencia (skinning features).

- *Accordion* <p:accordionPanel>: Es un componente contenedor con panel verticales apilados. Se puede seleccionar cuál o cuáles de ellos están desplegados (mostrando su contenido) en cada momento sin más que hacer clic en ellos. Cada panel se implementa con un componente Tab.

- *TabView* <p:tab>: Es un componente de panel usado como contenedor para agrupar contenido en pestañas (tab).

4.2.Vis.js

En la parte cliente se empleó la librería vis.js para la visualización de la información recibida del servidor Neo4j, como respuesta a la consulta Cypher, en forma de grafo.

Vis.js es una librería de visualización basada en navegador y dinámica. Está diseñada para ser fácil de usar, manejar grandes cantidades de datos dinámicos y habilitar la manipulación e interacción con los datos [26]. La librería consta de cinco tipos de componentes:

- *Network* (red): Para mostrar vistas dinámicas, organizables automáticamente y personalizables de redes.

- *Timeline* (línea temporal): Indicado para crear líneas temporales interactivas y totalmente personalizables con ítems y rangos.

- *Graph2d* (gráficos en 2 dimensiones): Para dibujar gráficos y diagramas de barras sobre una línea temporal y personalizarlos a nuestro gusto.

- *Grahp3d* (gráficos en 3 dimensiones): Para dibujar gráficos interactivos y animados en 3 dimensiones.

- *DataSet* (conjunto de datos): Orientado a manejar datos desestructurados. Entre las acciones válidas están añadir, actualizar y borrar datos, así como “escuchar” en busca de cambios en los datos.

En la parte cliente, se emplearon los componentes DataSet para guardar la información de los nodos y las relaciones y Network para crear una red y visualizarla en el portet.

4.3. Parte cliente

La parte cliente establece lo que el usuario ve y cómo lo ve. Se dividió en tres zonas, delimitadas por elementos HTML de tipo formulario <h:form>:

- Formulario “consola”: Incluye varios elementos del showcase de PrimeFaces como un “terminal”, donde se escriben las consultas Cypher y un “commandButton” para borrar la consola. Un elemento “focus” establece el foco en la consola cuando se carga el portlet y después de borrarla.

- Formulario “infografo”: Consta de un componente PrimeFaces accordionPanel”, que desplegado muestra el valor de los nodos y las relaciones visualizadas en el grafo, y de dos “commandButton”, uno para actualizar la información de nodos y relaciones con la devuelta por la última consulta Cypher válida y otro para actualizar la visualización del grafo.

- Formulario “grafo”: Es un elemento de HTML que define una sección dentro del documento XHTML donde se dibujará el grafo empleando una script de JavaScript que llamará a funciones de la librería vis.js, en concreto a los componentes Dataset y Network.

4.4.Parte servidor

La parte servidor contiene la funcionalidad del portlet, implementada en la clase terminalportlet.java. En esta, se incluye el manejador de eventos (commandHandler) del componente “terminal”, que se activa cuando se pulsa “Enter” y toma la entrada de teclado. Se trata dicha entrada como los comandos del intérprete de órdenes. Por simplificar, dichos comandos serán consultas válidas de Cypher o no.

Se envía la entrada a una función que realiza una petición al servidor Neo4j empleando las funciones para embeber Neo4j en aplicaciones Java [6]. Entonces, se tiene dos escenarios posibles:

- Si la consulta es válida y no hay problemas, el manejador devuelve a la consola un mensaje indicando que la consulta es válida, mientras que el servidor Neo4j devuelve un JSON en formato de Neo4j, que se trata para construir un JSON que la librería vis.js de visualización acepte.

- Si la consulta no es válida, salta una excepción que se trata enviando un mensaje a la consola avisando de que el comando lanzado en ella no era una consulta Cypher válida. Además, se devuelve el control a la consola para que el usuario pueda seguir lanzando consultas Cypher.

4.5.Portlet Firmament

En este apartado, se presentará el portlet desarrollado desde el punto de vista de la interfaz gráfica.

Como ya se comentó, el portlet consta de de 3 partes.

En la figura 8, se puede ver la parte superior del portlet, formada por una consola que hace la función de intérprete de órdenes. En este componente es donde se escriben las consultas Cypher que se lanzan al servidor Neo4j. Además, debajo de la consola hay un botón "Borrar" para limpiar la consola, es decir, hacer que desaparezcan los comandos ya ejecutados.

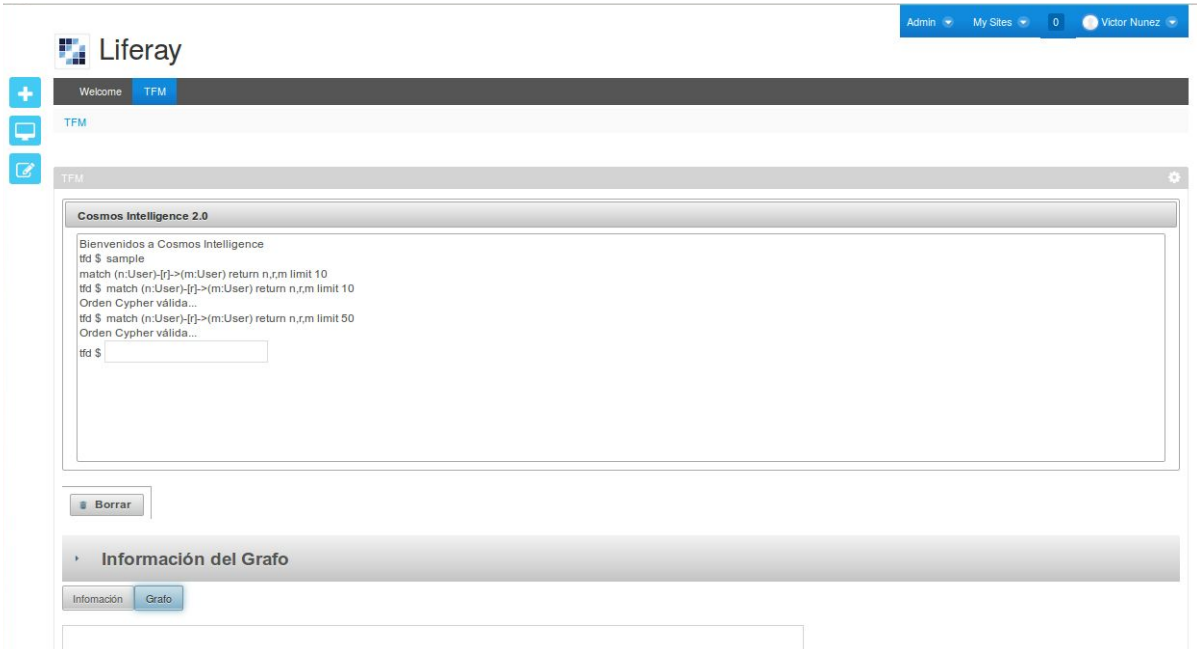

Figura 8. Vista general del portlet con la parte de consola arriba y la pestaña de información sin desplegar.

Debajo de la consola y el botón "Borrar", se sitúa la parte intermedia del portlet. Esta consta de una pestaña "Información del Grafo" que, al hacer clic sobre ella, muestra la información actualmente cargada de los nodos y de las relaciones. Adicionalmente, hay dos botones: "Información" sirve para cargar la información de nodos y relaciones de la última consulta válida y "Grafo" hace que se visualice la información cargada dibujando el grafo.

En la figura 9, se muestra con detalle la consola. Se puede apreciar un mensaje de bienvenida y el "prompt" para escribir las órdenes. Escribiendo el comando "sample" aparece una consulta de ejemplo para hacerse una idea de cómo hay que interrogar a la base de datos.

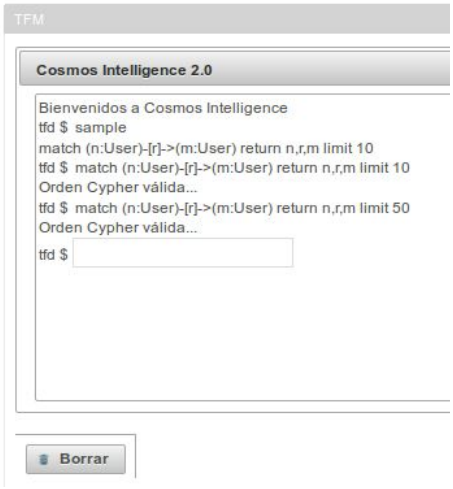

Figura 9. Detalle de la consola con el botón Borrar.

En la parte inferior del portlet se encuentra el área donde se dibuja el grafo cuando se pulsa el botón “Grafo”. En la figura 10, se puede ver el grafo resultante y la pestaña “Información del grafo”. Esta, aparece desplegada y mostrando las lista de nodos y relaciones cargados al pulsar el botón “Información”.

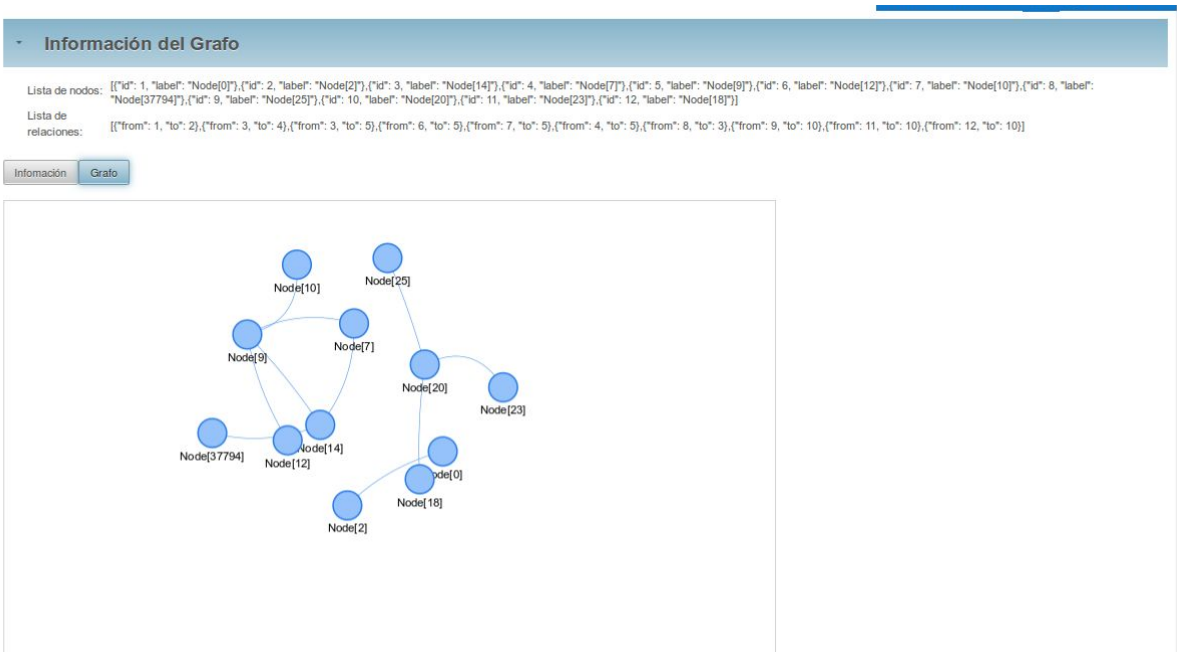

Figura 10. Vista del portlet con la pestaña de información del grafo desplegada y el grafo abajo.

Por último, en la figura 11 se muestra más de cerca el grafo resultante. Se observa que está compuesto de 12 nodos y 10 relaciones. Si se hace clic con el ratón fuera de los nodos y relaciones se puede mover de sitio el grafo en su totalidad. Con la rueda del ratón, se puede hacer más grande o más pequeño. Si se hace clic y se mantiene en los nodos, se puede cambiar su ubicación, con lo que el resto de nodos conectados o "relacionados" con él también alteran su posición.

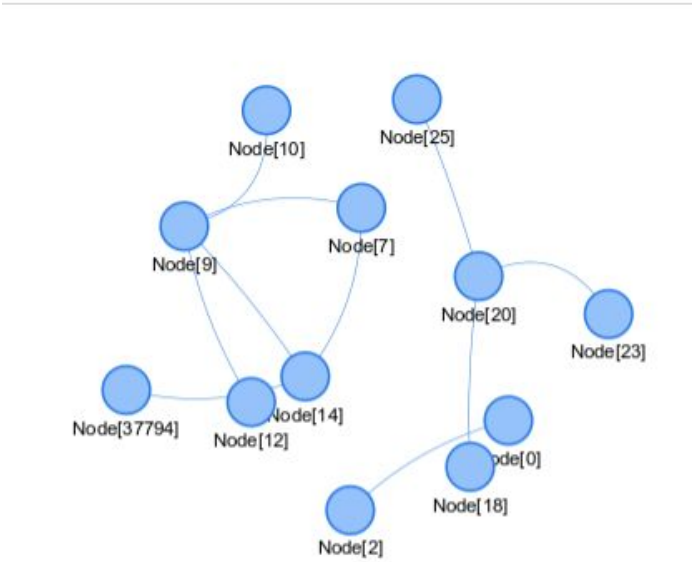

Figura 11. Detalle de la representación del grafo empleando la librería Vis.js.

5.Análisis de Rendimiento

En este apartado, se realizará una medición del rendimiento del portlet implementado en este trabajo. Con tal propósito, se ha tomado una consulta de prueba sencilla:

match (n:User)-[r]->(m:User) return n,r,m limit 10

en la cual se pregunta por los nodos tipo User unidos directamente por una relación. Esta consulta se ha ejecutado, para diferente número límite de resultados devueltos, en la misma máquina en la cual se ha desarrollado el portlet pero en dos escenarios distintos:

- sobre la aplicación web de Neo4j
- sobre el portlet desarrollado e incluido en una instancia Liferay Portal. Los datos se recogieron en la tabla 5.

La base de datos de grafo Neo4j de prueba consta de 240.275 nodos y 2.270.321 relaciones, lo que justifica el hecho de marcar un límite. El dominio de dicha base de datos es la red social Twitter.

Límite	Aplicación Web Neo4j	Portlet
10	1 s. (27 ms.)	1 s.
100	2 s. (129 ms.)	8 s.
300	6 s. (56 ms.)	27 s
400	15 s. (57 ms.)	69 s.
500	21 s. (503 ms.)	93 s.
600	28 s. (461 ms.)	116 s.
700	31 s. (279 ms.)	143 s.
800	34 s. (464 ms.)	169 s.
900	160 s. (527 ms.)	213 s.
1000	¿?	236 s.

Tabla 5. Comparativa de rendimiento entre la aplicación web Ne4j y el portlet.

En ambos casos, la información se recupera con gran celeridad: en menos de un 1 segundo para la aplicación web de Neo4j, mientras que para el portlet tarda algo más. Esto es así porque hay que realizar procesos adicionales antes de visualizarla: leer el resultado del servidor Neo4j (un objeto de tipo Result), guardarlo en un objeto de tipo StringBuilder y convertir el formato a un tipo JSON “entendible” por la librería vis.js en la parte servidor.

Este proceso es transparente para el usuario en la aplicación Web de Neo4j o no se requiere porque la información no sale del ámbito de Neo4j.

Por ejemplo, para recuperar el número de todos los nodos con la sencilla consulta Cypher

```
match (u) return count(u)
```

se tarda tan solo 2.503 ms., unos 2,5 s., para contar 240.275 nodos.

Para recuperar el número de todos los nodos con la consulta

```
match ()-[r]-() return count(r)
```

se tarda 17.578 ms., unos 17,5 s., para contar 2.270.321 relaciones.

De todos modos, donde realmente se encuentra el “cuello de botella” es en la parte de visualización del grafo. En ambos casos, la visualización no es inmediata. Para el caso de 500 resultados, hubo que esperar 21 segundos en la aplicación web de Neo4j mientras que 93 s con el portlet incluido en el portal Liferay.

Sin embargo, aún tardando más en el caso del portlet, la visualización es menos “pesada” para el sistema puesto que consume menos recursos una vez dibujada. En el caso de la aplicación web Neo4j, la visualización se vuelve inmanejable para cantidades elevadas de resultados: no se renderiza bien, cuesta desplazarse por la aplicación o mover nodos y relaciones haciendo clic en ellos o desplazar el grafo haciendo clic fuera del mismo y desplazando y, en general, se ralentiza el funcionamiento del equipo donde se ejecute.

Por último, el portlet admite mayor cantidad de resultados, es decir, es capaz de visualizar grafos con más nodos y relaciones aún con la desventaja de tardar más en dibujarse. La aplicación web de Neo4j es incapaz de dibujar grafos de más de 900 resultados y tiene problemas para hacerlo correctamente con tan solo 100 resultados.

En la figura 12, se puede observar los problema de renderizado de la aplicación web de Neo4j con tan solo 100 resultados.

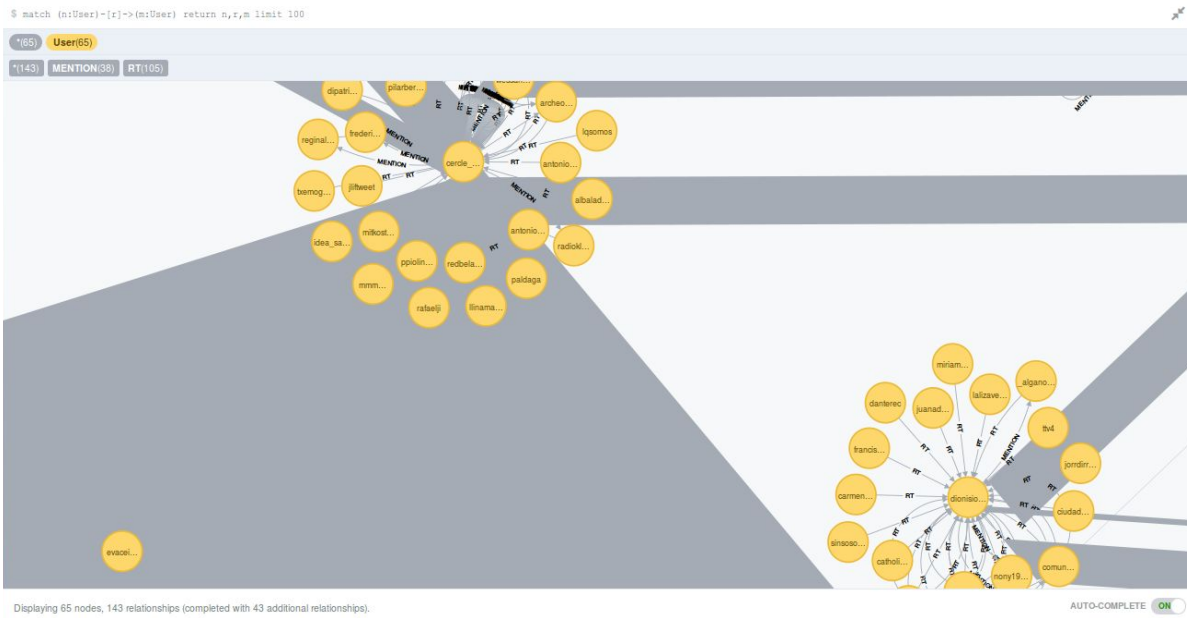

Figura 12. Problemas de renderizado de la aplicación web de Neo4j.

El portlet es capaz de dibujar grafos de 1000 resultados en menos de 4 minutos y se puede navegar por él sin problemas. En la figura 13, se muestra dicho grafo de 1000 resultados generado por el portlet sin aproximar la imagen.

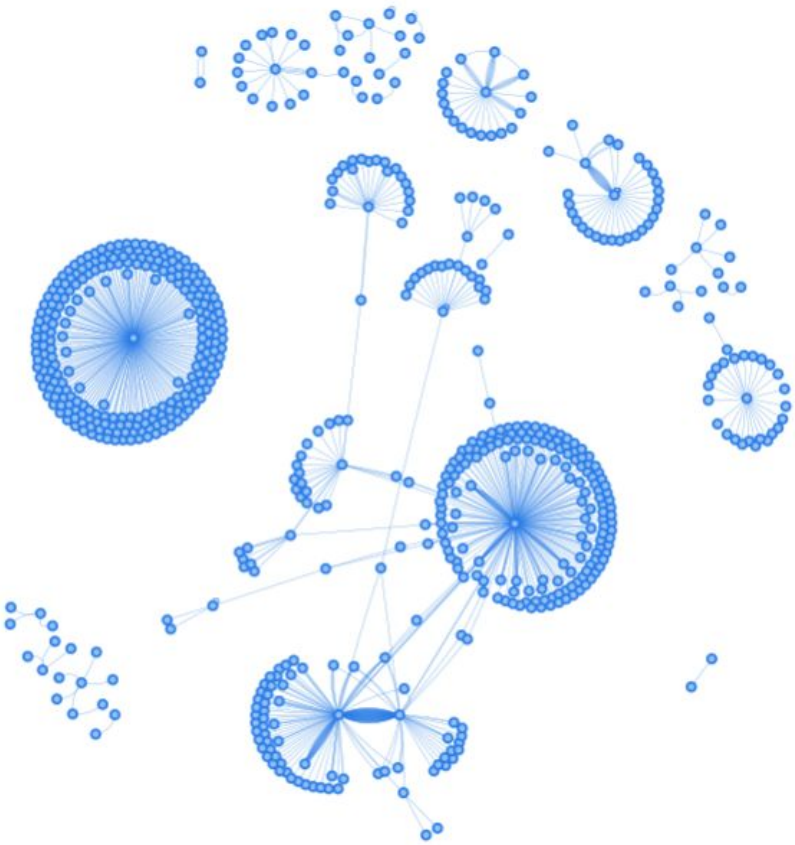

Figura 13. Grafo de 1000 resultados generado mediante el portlet.

Si se aproxima la imagen al grupo de nodos de la izquierda, se puede ver con mayor claridad una gran acumulación de nodos y relaciones en torno al nodo de “id” 165. En la figura 14, se observa dicho efecto.

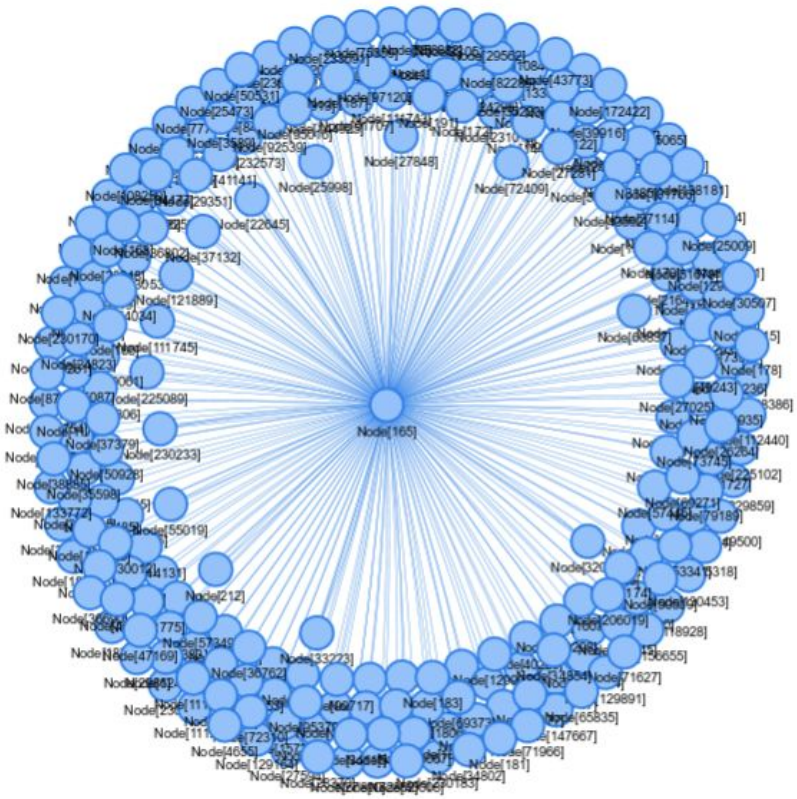

Figura 14. Detalle del grafo de resultados de 1000 resultados generado por el portlet.

En conclusión, para realizar consultas de tipo numérico o cálculos, la aplicación web de Neo4j es la única opción puesto que el portlet solo está a visualizar grafos. Si se pretende visualizar el grafo en toda o en gran parte de su magnitud, el portlet ofrece mejores prestaciones: es más liviano en cuanto a uso de rendimientos y se puede interactuar con él sin problemas.

6.Conclusiones

En este trabajo, se ha desarrollado un portlet que integra la consulta de la base de datos de grafo Neo4j y la visualización de la información recuperada en forma de grafo. Adicionalmente, se puede desplegar la información de las nodos y las relaciones.

Primero, se realiza una consulta Cypher. El siguiente paso, es cargar la información de los nodos y las relaciones pulsando el botón “Información”. Por último, se visualiza la información cargada pulsando el botón “Grafo”.

En cuanto a las ventajas aportadas por el portlet desarrollado, cabe destacar que:

- Se amplía el rango de comandos que se pueden efectuar desde el intérprete ya se introduce una capa entre el usuario y Neo4j.
- La visualización no tiene problemas de dibujado.
- Permite visualizar grafos con mayor número de resultados,
- La visualización, una vez dibujada, permite interactuar con ella de manera más ágil que la aplicación web de Neo4j.

En cuanto a las desventajas:

- Tarda más tiempo en dibujar el grafo.
- La interfaz gráfica tiene más potencial a desarrollar todavía.

En conclusión, el portlet integra varias tecnologías en una misma herramienta con el denominador común de Java. Esto le permite no limitarse a una sola tecnologia,como en el caso de la aplicación web de Neo4j, y poder añadir funcionalidad y mejorar el rendimiento.

La mejora en el rendimiento es de especial relevancia en la parte de visualización, el verdadero “lastre” de la aplicación web de Neo4j. Tanto el portlet como la aplicación web, devuelven la información en un tiempo aceptable pero el problema surge a la hora de visualizarla.

Como trabajo futuro, se podría mejorar la interfaz gráfica añadiendo más componentes de PrimeFaces y añadir más efectos en la visualización de la librería vis.js. También, se podría añadir más comandos al intérprete de órdenes para enriquecer el portlet con mayor funcionalidad. Además, se podría añadir la opción de guardar los datos devueltos en formato tabulado para su análisis en un herramienta de aprendizaje automático.

BIBLIOGRAFÍA

[1] Graph database - Wikipedia.
https://en.wikipedia.org/wiki/Graph_database

[2] “Lesson 8: Graph Databases”, William Vorhies.
http://www.datasciencecentral.com/profiles/blogs/lesson-8-graph-databases

[3] Autoritas Consulting.
http://www.autoritas.net/

[4] Social Intelligence - Wikipedia.
https://en.wikipedia.org/wiki/Social_intelligence

[5] Apache Lucene.
http://lucene.apache.org/

[6] “The Neo4j Manual v2.2.3” Chapter 35. Using Neo4j embedded in Java applications.
http://neo4j.com/docs/2.2.3/tutorials-java-embedded.html

[7] “Inteligencia Social de Negocio con Neo4j”, Andrés Ramos Pérez. Diciembre de 2015. Diploma de Especialización en Big Data. Trabajo Fin de Diploma.

[8] NoSQL - Wikipedia.
https://en.wikipedia.org/wiki/NoSQL

[9] “NoSQL Death to Relational Databases(?)”, Ben Scofield.
http://www.slideshare.net/bscofield/nosql-codemash-2010

[10] “Understanding Neo4j Scalability”, David Montag. Enero de 2013.
http://info.neo4j.com/rs/neotechnology/images/Understanding%20Neo4j%20Scalability%282%29.pdf

[11] “DB-Engines-Ranking - Trend of Graph DBMS Popularity”. Diciembre de 2015.
http://db-engines.com/en/ranking_trend/graph+dbms

[12] “What is a graph database?” Documentación para desarrolladores del sitio web oficial de Neo4j.
http://neo4j.com/developer/graph-database/

[13] Neo4j - Clientes.
http://neo4j.com/customers/

[14] Neo4j - Casos de uso.
http://neo4j.com/use-cases/

[15] ACID - Wikipedia.
https://en.wikipedia.org/wiki/ACID

[16] Neo4j - Lenguaje de consulta Cypher.
http://neo4j.com/developer/cypher-query-language/

[17] openCypher.
http://www.opencypher.org/

[18] Portlet - Wikipedia.
https://en.wikipedia.org/wiki/Portlet

[19] Web Portal - Wikipedia.
https://en.wikipedia.org/wiki/Web_portal

[20] Enterprise Portal - Wikipedia.
https://en.wikipedia.org/wiki/Enterprise_portal

[21] Liferay Portal - Wikipedia.
https://en.wikipedia.org/wiki/Enterprise_portal

[22] Liferay Portal 6.2 Developer’s Guide - Working with Liferay’s Developer Tools - Developing Apps with Liferay IDE.
https://www.liferay.com/es/documentation/liferay-portal/6.2/development/-/ai/developing-apps-with-liferay-ide-liferay-portal-6-2-dev-guide-02-en

[23] devrates.com - Reseñas sobre código abierto realizadas por usuarios reales.
http://devrates.com/stats/index

[24] PrimeFaces - Getting started.
http://www.primefaces.org/gettingStarted

[25] PrimeFaces Showcase
http://www.primefaces.org/showcase/

[26] Vis.js - Librería de visualización.
http://visjs.org/

Anéxo A. Código del cliente

```
<?xml version="1.0"?>
<f:view
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:c="http://java.sun.com/jsp/jstl/core"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:p="http://primefaces.org/ui"
  xmlns:ui="http://java.sun.com/jsf/facelets"
>
<h:head>
  <style type="text/css">
        #mynetwork {
              width: 1000px;
              height: 800px;
              border: 1px solid lightgray;
        }
   </style>
</h:head>
<h:body>

<h:form id="consola">
         <p:panel id="panelConsola" header="Cosmos Intelligence 2.0"
style="margin-bottom:10px">
        <p:focus for="terminal" />
        <p:terminal id="terminal" widgetVar="term"
commandHandler="#{terminalportlet.handleCommand}"
        welcomeMessage="Bienvenidos a Cosmos Intelligence" prompt="tfd $ "/>
        <p:commandButton type="button" value="Borrar" icon="ui-icon-trash"
              onclick="PF('term').clear(), PF('term').focus()" style="margin-top:5px" />
        </p:panel>
```

```
</h:form>

<h:form id="infografo">
        <p:accordionPanel id=”acordeon”>
                <p:tab title="Información del Grafo">
                        <h:panelGrid columns="2" celpadding="5">
                        <h:outputLabel id="nodeList" value="Lista de nodos: " />
                        <h:outputText id="input_nodes" value="#{terminalportlet.visnodes}" />
                        <h:outputLabel id="edgeList" value="Lista de relaciones: " />
                        <h:outputText id="input_edges" value="#{terminalportlet.visedges}" />
                        </h:panelGrid>
                </p:tab>
        </p:accordionPanel>

        <p:commandButton type="submit" id="showinfo" value="Información"
        update="_TFM_WAR_TFMportlet_:infografo:acordeon:input_nodes,
        _TFM_WAR_TFMportlet_:infografo:acordeon:input_edges"
                                styleClass="ui-priority-primary" style="margin-top:5px" />
        <p:commandButton type="submit" id="showgraph" value="Grafo"
        styleClass="ui-priority-primary" style="margin-top:5px" onstart="loadGraph()" />
  </h:form>

<h:form id="grafo">
        <div id="mynetwork"></div>
        <script type="text/javascript">
                function loadGraph(){
        var n =
document.getElementById("_TFM_WAR_TFMportlet_:infografo:j_idt6:input_nodes").textContent;
        var e =
document.getElementById("_TFM_WAR_TFMportlet_:infografo:j_idt6:input_edges").textContent;

                // parsear nodos y relaciones provenientes del servidor
```

```
var n_json = JSON.parse(n);
var e_json = JSON.parse(e);
// crear un array con los nodos
var nodes = new vis.DataSet(n_json);

// crear un array con las relaciones
var edges = new vis.DataSet(e_json);

// crear una red
var container = document.getElementById('mynetwork');

var data = {
        nodes: nodes,
        edges: edges
};

var options = {
        nodes: {
                shape: 'dot',
                size: 30,
                font: {
                        size: 24,
                                color: '#000'
                },
                borderWidth: 2
        },
        edges: {
                width: 1
        } ,
        physics: {
                forceAtlas2Based: {
                        gravitationalConstant: -26,
```

```
                                centralGravity: 0.005,
                                springLength: 230,
                                springConstant: 0.18
                        },
                        maxVelocity: 146,
                        solver: 'forceAtlas2Based',
                        timestep: 0.35,
                        stabilization: {
                                enabled:true,
                                iterations:1000,
                                updateInterval:25
                        }
                },
                interaction: {
                        tooltipDelay: 200,
                        hideEdgesOnDrag: true,
                        navigationButtons: true,
                                keyboard: true
                }
        };
        // crear una red en el elemento HTML con unos datos y unas opciones
        var network = new vis.Network(container, data, options);
        }
    </script>

</h:form>

</h:body>

</f:view>
```

Anéxo B. Código del servidor

```
package com.tfmportlet;

import com.liferay.util.bridges.mvc.MVCPortlet;

import java.util.Date;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;

import java.lang.StringBuilder;
import java.lang.String;

import org.neo4j.graphdb.QueryExecutionException;
import org.neo4j.graphdb.GraphDatabaseService;
import org.neo4j.graphdb.Result;
import org.neo4j.graphdb.Transaction;
import org.neo4j.graphdb.factory.GraphDatabaseFactory;

/**
* Portlet implementation class terminalportlet
* @author Víctor Núñez Monsálvez
*/

@ManagedBean
@SessionScoped
public class terminalportlet extends MVCPortlet {

public static String NEO4JPATH = "/home/victor/Descargas/bd-tfd/tfm_neo/neo.db/tuits.db";
```

```
private String visnodes;
private String visedges;

public String handleCommand(String command, String[] params) {

        switch(command){
        case "date":    return new Date().toString();
        case "greet":    if(params.length > 0)
                                        return "Hello " + params[0];
                                else
                                        return "Hello Stranger";
        case "help": return "Comandos válidos: date, greet, help, path, sample";
        case "path":
                                return NEO4JPATH;
        case "sample": return "match (n:User)-[r]->(m:User) return n,r,m limit 10";
        default:
                String res_request = request_neo4j( command, params);
                return res_request;
        }

        }

private static void registerShutdownHook( final GraphDatabaseService graphDb ){
        // Registers a shutdown hook for the Neo4j instance so that it
        // shuts down nicely when the VM exits (even if you "Ctrl-C" the
        // running application).
        Runtime.getRuntime().addShutdownHook( new Thread(){
                @Override
                public void run(){
                        graphDb.shutdown();
                }
```

```
        } );
}

public String request_neo4j(String command, String[] params){
        //Aqui se recibe la orden Cypher y se envia al servidor
                GraphDatabaseService graphDb = new
GraphDatabaseFactory().newEmbeddedDatabase( NEO4JPATH );

        registerShutdownHook( graphDb );

        //Para almacenar el result de neo4j como un String
        String rows = "";
        //Para guardar los nodos como pares {String Nodo[num_neo4j], int id_mio}
        Map<String, Object> nodos = new HashMap<String, Object>();
        //Map<String, Map> relaciones = new HashMap<String,Map>();

        //Crear json para nodos y relaciones
        StringBuilder gn = new StringBuilder();
        gn.append("[");
        StringBuilder ge = new StringBuilder();
        ge.append("[");

        // Inicializar identificador de nodos
        int id_node = 1;

        // Array de Strings donde se guardará el label de los nodos de cada relación
        String rel_side = "from";

        // Construir la orden Cypher que se ejecuta en el servidor Neo4j
        String orden = command;
        for (int i = 0; i < params.length; i++){
                orden += " " + params[i];
```

```
}
System.out.println( orden );

try (Transaction ignored = graphDb.beginTx();
        Result result = graphDb.execute( orden ) )
{
// iterar por filas para construir el json de nodos (gn) y relaciones (ge) al mismo
// tiempo que se lee el resultado obtenido de la consulta Cypher.
while (result.hasNext()) {

        Map<String, Object> row = result.next();
        for (Entry<String, Object> column : row.entrySet()){

        //String clave = column.getKey();
        String valor = column.getValue().toString();
        rows += column.getKey() + ": " + column.getValue() + "; ";
        if (valor.contains("Node")){

                if(nodos.isEmpty()){
                // Si hay nodos, se añade el nodo
                nodos.put(valor, id_node);
                // Añadir el nodo con id para visjs al json de nodes
                gn.append("{\"id\": ").append(id_node);
                gn.append(", \"label\": \"").append(valor).append("\"},");

                // Construir la relación
                if (rel_side == "from"){
                        ge.append("{\"from\": ").append(id_node).append(", \"to\": ");
                        rel_side = "to";
                }else{
                        ge.append(id_node).append("},");
                        rel_side = "from";
                }
```

```
// Incrementar id en 1
id_node = id_node +1;

}else if(nodos.containsKey(valor) != true){
        // Si hay nodos, se comprueba si ya está.
        // Si no está, se añade
        nodos.put(valor, id_node);
        // Añadir el nodo con id para visjs al json de edges
        gn.append("{\"id\": ").append(id_node);
        gn.append(", \"label\": \"").append(valor).append("\"},");

         // Construir la relación
        if (rel_side == "from"){
        ge.append("{\"from\": ").append(id_node).append(", \"to\": ");
        rel_side = "to";
        }else{
        ge.append(id_node).append("},");
        rel_side = "from";
        }

        // Incrementar id en 1
        id_node = id_node +1;

}else{
        // Si el nodo ya está, no se añade nada
        //pero si que se construye la relación
        // Construir la relación
        if (rel_side == "from"){
        //get(Object key): Returns the value to which
        //the specified key is mapped,
        //or null if this map contains no mapping for the key.
```

```
                        // nodos.get(valor) devuelve el id del nodo para vis.js
                        ge.append("{\"from\": ").append(nodos.get(valor)).append(", \"to\": ");
                                rel_side = "to";
                        }else{
                                ge.append(nodos.get(valor)).append("},");
                                rel_side = "from";
                        }
                }

        } else if (column.getValue().toString().contains("Relationship")){
        // Si es una relación no se hace nada, sino cuando se llegue a los nodos
        }
        // gn y ge se construyen al finalizar la lectura y tratamiento de
        // cada line del result de neo4j

}
// Cambio de línea en rows para guardar la siguiente fila del resultado
rows += "\n";

}

  //Añadir el final a los jsones
  gn.append("]");
  ge.append("]");

  //Eliminar la última coma antes de cerrar los corchetes ya que
  //no hay más nodos ni relaciones que añadir
  int start_comma = gn.lastIndexOf("},]");
  int end_comma = gn.length();
  gn.replace(start_comma, end_comma, "}]");

  start_comma = ge.lastIndexOf("},]");
```

```
            end_comma = ge.length();
            ge.replace(start_comma, end_comma, "}]");

            //Limpiar los jsones de "Node[" y "]]"
            System.out.println("String creado al leer el result de Neo4j...");
            System.out.println(rows);

            System.out.println("JSONs generados para nodes y edges...");
            System.out.println(gn.toString());
            System.out.println(ge.toString());

            // Modificar el valor de las variable privadas visnodes y visedges,
            // a partir de las cuales se dibujará el grafo en el cliente,
            // mediante un setter
            setVisnodes(gn.toString());
            setVisedges(ge.toString());


        ignored.close();
        graphDb.shutdown();

        } catch(QueryExecutionException e){
                graphDb.shutdown();
                return "Orden Cypher no válida...";
        }

        return "Orden Cypher válida...";

}

 public String getVisnodes(){
         // getter de visnodes
         return visnodes;
```

```
    }

    public void setVisnodes( String visnodes){
            // setter de visnodes
            this.visnodes = visnodes;
            System.out.println("VisNodes salvado...");
    }

    public String getVisedges(){
            // getter de visedges
            return visedges;
    }

    public void setVisedges(String visedges){
            // setter de visedges
            this.visedges = visedges;
            System.out.println("VisEdges salvado...");

    }


}
```