UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

SISTEMA DE CONTROL DE ACCESO PARA UN ENDPOINT

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERA CIVIL EN COMPUTACIÓN

DANIELA FRANCESCA AHUMADA OPPICI

PROFESOR GUÍA:
CLAUDIO GUTIÉRREZ GALLARDO

MIEMBROS DE LA COMISIÓN:
ALEXANDRE BERGEL
SERGIO OCHOA DELORENZI

SANTIAGO DE CHILE
2015

Resumen

Un *endpoint* es una aplicación similar a un servicio web que permite consultas estructuradas. Estas aplicaciones trabajan con el lenguaje de consulta SPARQL para consultar los datos y entender su semántica. Hoy día la mayoría de estas aplicaciones no tiene un sistema de seguridad que permita restringir el acceso de usuarios que tienen permisos para modificar estos datos. Este problema fue tomado como proyecto de memoria, y se implementó una solución genérica que permitire a los *endpoints* hoy existentes, agregar seguridad y administrar roles y usuarios mediante una interfaz gráfica.

La implementación de esta solución consistió en tres grandes módulos. La primera parte fue una investigación del estado del arte, donde se estudió lo que otros académicos han investigado sobre el tema. Aquí se contactó a dos grupos de investigadores que han estado involucrados en la Web Semántica y han desarrollado soluciones parciales a este problema. La segunda, consistió en el diseño de la solución, donde se tomó varios conceptos encontrados en la etapa anterior, y en la implementación de ésta. Finalmente, se validó la solución en términos de usabilidad, aplicando dos métodos (Observación y Cuestionario de Usuario Final), y también en términos de rendimiento.

La aplicación mostró resolver el problema planteado. Quedó a disposición de la comunidad con código abierto y licencia Atribución 4.0 Internacional de Creative Commons.

A mi familia.

Agradecimientos

Dedico el presente trabajo a mi familia por el constante apoyo que me brindaron durante toda la carrera.

Quiero agradecer a Claudio Gutiérrez, por guiarme en este proyecto que requirió una gran investigación al comienzo.

Además quiero agradecer a Acepta, la empresa en la cual he trabajado desde la primera práctica. Esta compañía me guió en el mundo de la computación y me ayudó a comprender que esta carrera era la adecuada para mi.

Tabla de Contenido

Capítulo 1

Introducción

Hoy en día la Web es el espacio de información más grande que poseemos. Además es una fuente a la cual un gran cantidad de personas tiene acceso. Las enciclopedias son algo que los niños de hoy casi no conocen, ya que Google les contesta mucho más y de manera mucho más rápida.

Dado que tenemos esta gran fuente de información a nuestra disposición, ¿No sería bueno que fuera consultable?, es decir, que pudiéramos hacerle preguntas un poco más complejas que las que responde Google. Los buscadores que todos conocen, son buscadores que buscan palabras en un texto o documento, y la relación entre los resultados es esencialmente el tener dentro de su contenido la palabra buscada.

¿Cómo se podría hacer la Web consultable? Como comienzo se debería tener un estándar para publicar datos en la Web. Si un conjunto de datos sigue un patrón es mucho más eficiente la búsqueda sobre ellos.

La Web Semántica trata de solucionar este problema. Ésta es una ampliación de la Web, la cual busca realizar un filtrado automático pero preciso de la información. Para ello dispone de lenguajes para codificar datos y su semántica, como *Resource Description Framework*[1] (RDF) y *Web Ontology Language*[2] (OWL). Estas tecnologías se combinan para aportar descripciones explícitas de los recursos de la Web (catálogos, formularios, mapas, etc.).

Las tecnologías recién mencionadas, RDF y OWL, permiten describir, conectar o relacionar datos. Si uno tiene un conjunto de datos en RDF por ejemplo, serían de más utilidad si se tuviera un lenguaje para consultarlos. De ahí nació SPARQL[3], un lenguaje para consultar RDF. Luego, surgieron los *endpoints*, un servicio que permite administrar una base de datos

[1] `http://www.w3.org/RDF/` Accesado 1 September 2014
[2] `http://www.webopedia.com/TERM/O/Ontology_Web_Language.html` Accesado 1 September 2014
[3] `http://www.w3.org/TR/rdf-sparql-protocol/` Accesado 1 September 2014

de grafos, que permitieron que estos datos estuvieran disponibles en la Web y además fueran consultables.

Un *endpoint* permite el acceso a un conjunto de datos, donde no sólo se puede consultar si no que también modificar estos datos. Dado que también permite la intervención de los datos, existe la necesidad de protegerlos de usuarios malintencionados. El objetivo de este proyecto de memoria es buscar una solución que permita mejorar la seguridad en el acceso a los *endpoints*, entregando un mecanismo de control de acceso.

Este documento describe los resultados de una previa investigación y el desarrollo de un proyecto que resuelve uno de los problemas planteados por académicos que estudian la Web Semántica. Éste problema busca cómo controlar el acceso a los datos de un *endpoint*.[4]

El texto está organizado de la siguiente forma: Luego de hacer una breve introducción a los conceptos relevantes del área en la cual se trabajará, se detallarán los pasos que se siguieron para cumplir el objetivo planteado. El desarrollo de este proyecto siguió el estándar de proyectos informáticos. Se comenzó por investigar sobre el tema y averiguar si existe gente en otras partes del mundo haciendo lo mismo (Sección 3). Al ver que el mismo proyecto no había sido realizado por nadie aún se construyó el modelo y la arquitectura de la solución (Sección 4.1). Antes de finalizar se desarrolló la solución (Sección 4.2) y se validó. Las validaciones realizadas fueron a nivel de usabilidad (Sección 4.4) y a nivel de *performance* (Sección 4.5). Finalmente se encuentra la conclusión del proyecto.

1.1. Problema a Abordar

Un *endpoint*, como se mencionó brevemente en la introducción es un servicio que permite administrar una base de datos RDF. Esta administración incluye agregar, modificar y eliminar grafos. Estas acciones se pueden hacer mediante una interfaz gráfica que proporciona cada uno de los *endpoint* o directamente por *requests* HTTP. Hoy en día existen varias implementaciones, como por ejemplo Fuseki y Virtuoso, pero esencialmente tienen las mismas funcionalidades.

Como se mencionó anteriormente, hoy en día los *endpoints* permiten consultar y modificar algunas fuentes de datos interesantes. El problema que tienen las implementaciones de SPARQL *endpoints* es que si bien proveen mecanismos de control, estos mecanismos no permiten hacer un filtro para que los usuarios con ciertos permisos puedan modificar, y el resto que no los tiene, pueda sólo leer la información. Si estos datos expuestos en la Web son suficientemente interesantes, existe la necesidad de protegerlos de los usuarios mal intencionados. A partir de esta necesidad nació este proyecto, que busca proveerle a los *endpoints* ya

[4]La aplicación está disponible en `http://54.68.113.139`, usuario genérico: usuario-g, clave:1234. Cualquier consulta sobre el acceso a la aplicación, escribir a danif88@gmail.com

existentes un mecanismo de control de acceso más flexible que el que tienen hoy en día.

La complejidad de este problema radica en lograr una solución modular, de manera que el que quiera agregar este sistema de control de acceso en un *endpoint*, no tenga que hacer modificaciones dentro de su implementación. Además, se tiene que considerar que la integración debe afectar lo menos posible la actual interacción de los usuarios con el *endpoint*, para no perder usuarios. Por lo tanto, la integración de esta solución a cualquier *endpoint* debe ser casi transparente.

1.2. Objetivos

1.2.1. Objetivo General

Resolver el problema de control de acceso que hoy en día tienen los *endpoints*, proveyendo niveles de acceso. Para cumplir este objetivo se buscará diseñar y desarrollar una aplicación genérica que permita controlar el acceso (inserción, edición y borrado) a los datos de un *endpoint* para usuarios con distintos roles.

1.2.2. Objetivos Específicos

Dado que lo más importante es proteger la data de las modificaciones no autorizadas, se optó por dividir a los usuarios en cuatro grupos: los que sólo pueden consultar, los que pueden modificar grafos, los que pueden subir grafos y los super-usuarios. Para cumplir con el objetivo recién planteado se debe haber cumplido los siguientes sub-objetivos:

- Revisión de *endpoints*. Determinar un *endpoint* sobre el cual se trabajará.
- Estudio de los modelos de control acceso (MAC, DAC, RBAC). Escoger un modelo de control de acceso. Analizar la posibilidad de aplicar, para el caso de *enpoints* basados en REST, el manejo de acceso ya existente para servicios Web Restful.
- Diseño de la solución.
- Implementación del módulo que se comunicará con el *endpoint* utilizando el modelo de control acceso seleccionado.
- Validación: Evaluación de Usabilidad.
- Validación: Pruebas de Estrés.

1.2.3. Entregables

1. Aplicación que implemente el módulo de control de acceso [5].
2. Documento de memoria.

[5]La aplicación está disponible en `http://54.68.113.139`, usuario genérico: usuario-g, clave:1234. Cualquier consulta sobre el acceso a la aplicación, escribir a danif88@gmail.com

Capítulo 2

Conceptos Básicos

2.1. RDF(*Resource Description Framework*)

El modelo de datos RDF fue diseñado para compartir y reutilizar información fácilmente. Los vocabularios RDF (conocidos como ontologías) son colecciones de tripletas RDF que pueden ser usadas para describir tanto el esquema como las instancias. Por ejemplo, FOAF[1] es un vocabulario conocido que es usado para describir a la gente y sus relaciones sociales en la Web. Los vocabularios están usualmente ubicados en *namespaces* comunes. Por simplicidad se utilizan prefijos para hacer referencia a ellos. Por ejemplo:

```
foaf:http://xnlns.com/foaf/0.1/
```

La data RDF puede ser codificada usando distintos formatos (por ejemplo, Notation3(N3)[2], RDF/XML[3] o JSON-LD[4]). En este proyecto se utilizará TriG[5], una extensión de N3, que utiliza '{' para agrupar tripletas en grafos identificados por IRIs[6]. Una tripleta RDF se puede definir formalmente como sigue:

Definición 1 (Tripleta RDF): Una tripleta RDF es representada por una tupla $\langle$ S,P,O $\rangle \in (I \cup B \cup L) \times I \times (I \cup B \cup L)$, donde S es el sujeto, P el predicado y O el objeto, e I, B, L son utilizados para representar IRIs, nodos blancos y literales respectivamente.
Ejemplo:
<http://dbpedia.org/resource/Alan_Turing>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://dbpedia.org/ontology/Scientist>

[1] `http://xmlns.com/foaf/spec/` Accesado 1 September 2014
[2] `http://www.w3.org/TeamSubmission/n3/` Accesado 1 September 2014
[3] `http://www.w3.org/TR/rdf-syntax-grammar/` Accesado 1 September 2014
[4] `http://www.w3.org/TR/json-ld/` Accesado 1 September 2014
[5] `http://www.w3.org/2010/01/Turtle/Trig/` Accesado 1 September 2014
[6] `http://www.w3.org/International/O-URL-and-ident.html` Accesado 1 September 2014

Definición 2 (Grafo RDF): Siguiendo la definición de una tripleta RDF, un grafo RDF G consiste en un conjunto de tripletas. El universo de G, es el conjunto de elementos en (I ∪ B ∪ L) que están en las tripletas de G y en el vocabulario de G.

Definición 3 (Dataset RDF): Dado un grafo G RDF, un dataset RDF consiste en un conjunto de grafos

$$\{G, (U_1, G_1), ..., (U_n, G_n)\}$$

, con exactamente un grafo default posiblemente vacío; y uno o más *named graphs*, que consisten en un par ⟨ nombre, grafo ⟩, donde el nombre ∈ (I ∪ B).

RDF es un modelo estándar para intercambio de información en la Web. RDF tiene características que facilitan la combinación de la información aun cuando los esquemas subyacentes difieran, y soporta especialmente la evolución de los esquemas a lo largo del tiempo sin requerir cambios mayores.

2.2. SPARQL(*SPARQL Protocol and RDF Query Language*)

SPARQL es un lenguaje de consultas para RDF, es decir, un lenguaje de consulta para bases de datos, que permite obtener y manipular información almacenada en formato RDF. Este lenguaje fue hecho por el RDF *Data Access Working Group* (DAWG) de la *World Wide Web Consortium*(W3C), y es reconocido como una de las tecnologías claves de la Web Semántica. El 2008, se convirtió en una recomendación oficial de la W3C. SPARQL permite hacer calce de patrones con tripletas y una amplia gama de operadores sobre ellos. Las consultas SPARQL retornan soluciones basadas en *pattern matching*. Patrones para triples son triples que pueden tener variables en la posición del sujeto, del predicado y del objeto. Múltiples patrones de triples en conjunto, conocido como patrón para grafos son usados para hacer un *match* de un subgrafo. Luego existen diversos operadores sobre patrones, como conjunción, unión, diferencia, etc.

Definición 4 (SPARQL *query*): Se asumen tres conjuntos infinitos disjuntos I, B, L (IRIs, nodos blancos y literales respectivamente). Un termino RDF es un elemento en el conjunto T = (I ∪ B ∪ L ∪ V). Una tripleta RDF es una tupía ⟨ S,P,O ⟩ ∈ (I ∪ B) × I × T, donde S es el sujeto, P el predicado y O el objeto. Un grafo RDF es un conjunto de tripletas. Adicionalmente, se asume la existencia the un conjunto infinito V de variables disjuntas de T. Denotaremos var(α) el conjunto de variables que pertenecen a la estructura α. Una *query* SPARQL es una expresión de la forma Q = (W,P,G), donde W es un conjunto de variables de proyección, P es el patrón del grafo y G es el grafo. Se asume que cada *query* Q satisface que cada variable perteneciente a W también pertenece a P. Un patrón de grafo es definido repulsivamente de la siguiente forma:

- Una tupla de (I ∪ L ∪ V)×(I ∪ V)×(I ∪ L ∪ V) es un patrón de grafo llamado patrón de triples.
- Si P y R son patrones de grafos entonces { P AND R }, { P UNION R }, { P OPT R }, { P MINUS R } son patrones de grafos.
- Si P es un patrón de grafo y C es un filtro entonces { P FILTRO C } es un patrón de grafo. Un filtro se define recursivamente como sigue:
  - Si ?X, ?Y ∈ V y c ∈ (I ∪ L) entonces ?X = c, ?X = ?Y y la unión de los ?X son filtros atómicos.
  - Si C y D son filtros entonces !C, (C ||D) y (C && D) son filtros complejos.

2.3. *Endpoint*

Un *enpoint* SPARQL es un software que permite a los usuarios hacer consultas sobre una base de datos mediante el lenguaje SPARQL. Los resultados generalmente son devueltos en uno o más formatos que son procesables por máquinas. Un *enpoint* SPARQL es un *conformant SPARQL Protocol service*. Según la W3C[7] este tipo de servicio debe:

- Implementar la interfaz de SparqlQuery, que contiene una operación, query, que es utilizada para transmitir un *string* de una *query* SPARQL y, opcionalmente, una descripción de un *dataset* RDF.
- Implementar HTTP, SOAP, o ambos HTTP y SOAP para comunicar la query con la interfaz SPARQL.
- La implementación del HTTP y del SOAP está normado por un xsd[8] y un wsdl[9].
- Debe ser consistente con ciertas normativas de seguridad[10]. Un *SPARQL Protocol service* puede:
  - Hacer *requests* HTTP desde otro servidor origen a favor de sus clientes.
  - Ser utilizado como vector para enviar *requests* a otros sitios o servicios.

[7] `http://www.w3.org/TR/rdf-sparql-protocol/` Accesado 1 September 2014
[8] `http://www.w3.org/TR/xmlschema11-1/` Accesado 1 September 2014
[9] `http://www.w3.org/TR/2001/NOTE-wsdl-20010315` Accesado 1 September 2014
[10] `http://www.w3.org/TR/rdf-sparql-protocol/#policy` Accesado 1 September 2014

Por lo tanto, estos servicios tienen que actuar como *proxies* para terceros. Para cumplir con esto, los servicios deben:

- Poner restricciones en los recursos que pueden ser recuperados
- Deben loggear las *requests* de los clientes de manera que sea más sencillo seguirles el paso.
- Deben detectar las *queries* inseguras, poner un tiempo o memoria limite para las *queries*, o imponer otras restricciones para reducir la vulnerabilidad.

2.4. Modelos de Control de Acceso

En la Web, el importante controlar el que un usuario cualquiera modifique los datos, dejando inconsistencias en ellos o simplemente datos incorrectos. Por esta razón existen los modelos de control de acceso. Se distinguen tres grandes modelos de control de acceso conocidos por las siglas: MAC, DAC, RBAC. En el modelo MAC (*Mandatory Access Control*) existe un administrador que le da acceso a los usuarios. Sólo el administrador puede modificar la seguridad de los objetos o los permisos de los usuarios. A diferencia del modelo MAC, en el modelo DAC (*Discretionary Access Control*) el acceso a los recursos está basado en la identidad de los usuarios. Un usuario tiene permiso de acceso a un recurso por estar en una Lista de Control de Acceso asociada a ese recurso. Este modelo está basado en que si una persona es dueña de un objeto en DAC, puede darle permisos a otros usuarios los que pueden acceder. Por último, el RBAC (*Role-Based Access Control*) está basado en la asignación de roles al usuario. Al igual que en MAC, existe un administrador, que asigna al usuario un rol que tiene ciertos derechos y privilegios.

Capítulo 3

Trabajos Relacionados

Para este trabajo se leyeron los trabajos relacionados con seguridad en *Linked Data*. De esa revision bibliográfica se consolidaron conceptos como Modelos de Control de Acceso, *Endpoints* y *Web Services*, que se explican en la sección 3.1, y las relaciones que tienen los principales artículos 3.2 revisados para el desarrollo de la solución, así como los proyectos similares ya implementados 3.3.

3.1. Conceptos

A continuación se exponen los conceptos consolidados durante la revisión bibliográfica. Estos conceptos fueron muy relevantes durante el desarrollo del proyecto.

3.1.1. Modelos de Control de Acceso

El modelo de control de acceso es mandatorio para este proyecto. Si bien existen varios modelos, no todos ellos pueden ser aplicados al proyecto, ya que cada uno tiene sus pros y sus contras. En [2] se analizan los beneficios y problemas de tres grandes modelos, *Discretionary Access Control* (DAC), *Mandatory Access Control* (MAC), *Role-Based Access Control* (RBAC). Luego, plantean que RBAC ha llegado a ser un modelo bastante completo que incluye integridad y confidencialidad. Si bien se plantea RBAC como el modelo más actualizado de los tres, necesita un administrador de roles. Si se tiene en cuenta que cualquier persona en la Web puede querer tener acceso a un *endpoint*, la administración de los roles se vuelve cada vez más compleja. Por lo tanto, este modelo no es lo que se busca.

Para un *endpoint*, lo más fácil es que el que sube información sea dueño de ésta, y sea él el que le de acceso al resto a esa información a otras personas. Así es como funciona el modelo DAC. En [8] explica cómo se puede aplicar DAC tanto en el modelo entidad relación como en el modelo de árboles. Luego, expone con más detalle cómo se puede aplicar DAC en

RDF, donde parte explicando el modelo RDF, sigue con la asignación de permisos y las reglas de derivación, y termina definiendo las funciones de 'GRANT' y 'REVOKE' para SPARQL. Las consultas en un *endpoint*, dependiendo de la cantidad de datos pueden tardar bastante en retornar los resultados. Esto también ocurre en las bases de datos, y esta es una de las razones de que existan vistas en las bases de datos.

En [5] se plantea un modelo basado en vistas, pensando en que no toda la gente debiera tener acceso a la misma información. El modelo que plantea este *paper* no se preocupa de los *updates* en los datos que es lo nos concierne en este proyecto. Sin embargo, no es malo incluir un acceso a vistas dentro del modelo, de manera que los usuarios puedan hacer consultas sobre el set de datos que les interesa y no tarden tanto tiempo en obtener los resultados.

3.1.2. *Endpoints*

Hoy en día existen varios softwares para implementar *endpoints* en la Web. Uno de los más conocidos es Virtuoso. Virtuoso es el *endpoint* que más ha avanzado en el área de control de acceso, dado que ya provee un sistema para manejar los permisos sobre los grafos. El modelo de control de acceso que posee Virtuoso utiliza como base el modelo de control de acceso que poseen las bases de datos hoy en día. Si bien, esto que ya está desarrollado es similar a lo que se quiere implementar en este proyecto, es algo que depende de la estructura de Virtuoso, es decir, no es modular y por lo tanto no es fácil reutilizarlo para aplicarlo a otros *endpoints*. La idea de este proyecto es hacer un módulo que actúe como un *middleware* para cualquier *engine* que implemente el protocolo SPARQL y que se pueda ser utilizado con cualquier interfaz de consulta. Como motivación se pensaron los siguientes posibles usos:

- El que quiera crear un nuevo *engine* de SPARQL, no se tendrá que preocupar de la capa de permisos porque podrá ponerlo detrás de este *frontend*.
- Varios *engines* de SPARQL distintos podrían ser accedidos bajo el mismo *frontend*, con los mismos usuarios y roles.

Si bien se va a hacer un módulo, se trabajará sobre un solo *endpoint* para hacer las pruebas. Se trabajará sobre el *endpoint* Fuseki, que es un servidor SPARQL *open source*. Este servidor provee una interacción REST con la data RDF usando el protocolo SPARQL sobre HTTP. Fuseki utiliza Apache Jena, que es un *framework open source* de Java. Apache Jena sirve para construir aplicaciones de Web Semántica y *Linked Data*.

3.1.3. Web Services

Fuseki como se dijo anteriormente provee una interacción mediante un servicio Web REST, y no es el único *endpoint* que provee una interacción mediante un servicio Web. Por esta razón se pensó que tal vez se podría reutilizar las herramientas ya existentes para control de acceso en servicios Web. En [1] se menciona XACML(*Extensible Access Control Markup Language*) como herramienta de control de acceso para servicios Web. XACML es un lenguaje

de políticas de control de acceso implementado en XML, que describe cómo se deben evaluar las *requests* de acuerdo a las reglas definidas en las políticas. Es principalmente un sistema *Attribute Based Access Control*, donde los atributos asociados a un usuario, acción o recurso son los inputs en el momento de la toma de decisiones de acceso a un recurso específico. A pesar de que XACML es un lenguaje bastante completo para manejar los accesos, es un lenguaje muy grande para lo que se busca hacer en este proyecto.

3.2. Discusión Bibliográfica

A continuación se presenta un breve resumen de los principales trabajos escogidos y sus relaciones con el trabajo de esta memoria.

[1] Claudio Agostino, Ernesto Damiano, Sabrina De Capitani.
Pierangela Samarati.
"A web service architecture for enforcing access control policies"

Resumen: Da a conocer un modelo de políticas de acceso para SOAP. La arquitectura del sistema que plantea el *paper* es una arquitectura formada por tres módulos. Primero está el módulo que interactúa con el servicio(*The Policy Decision Point*), segundo está el módulo que evalúa las políticas(*The Policy Evaluation Point*) y por último el módulo que decide que políticas son aplicables en cada caso(*The Policy Administration Point*).

Discusión: Si bien es interesante cómo se manejan las políticas de acceso en un *Web service* SOAP, para nuestro caso esta maquinaria es algo que hila muy fino. En nuestro caso se busca, a grandes rasgos, dividir a los usuarios en dos para evitar que todos modifiquen la información de un *endpoint*.

[2] Ryan Ausanka-Crues.
"Methods for Access Control: Advances and Limitations"

Resumen: En el paper el autor analiza cuatro modelos distintos para manejar los accesos a una determinada información. Los modelos analizados son: *Discretionary Access Control* (DAC), *Mandatory Access Control* (MAC), *Role-Based Access Control* (RBAC), *Domain Type Enforcement* (DTE). Como en todo análisis se dan a conocer los beneficios y problemas que tienen cada uno de estos modelos. MAC y DAC tuvieron sus primeras implementaciones en los 70's. Luego los investigadores aprendieron sobre la complejidad de mantener políticas de seguridad y RBAC ha llegado a ser un modelo bastante bueno, que incluye integridad y confidencialidad.

Discusión: Si bien se plantea que RBAC es el modelo más actualizado dentro de los tres grandes modelos (RBAC, MAC, DAC), es un modelo que necesita de un administrador de roles. Si uno tiene en cuenta que cualquier persona en la Web puede querer tener acceso a un conjunto de datos, la administración de los roles se vuelve cada vez más compleja.

[3] Luca Costabello, Serena Villata, Oscar Rodriguez Rocha, Fabien Gandon.
"Access Control for HTTP Operations on Linked Data"

Resumen: Los autores explican el funcionamiento del sistema Shi3ld, construido para manejar el control de acceso en *Linked Data*. Es un modelo que está construido sobre la noción de *Named Graphs*. El sistema se plantea como un filtro genérico para *enpoints* SPARQL, donde el *endpoint* no tiene que sufrir alteraciones al momento de la integración. Este filtro al momento de aplicar las políticas de acceso lo que hace, de ser necesario, modificar la consulta de manera que se ajuste a los permisos que tiene el usuario.

Discusión: El programa Shi3ld es un programa que se parece a lo que se quiere implementar pero tiene un enfoque distinto que se explica en la sección 3.3.

[4] David F. Ferraiolo, Janet A. Cugini, D. Richard Kuhn.
"Role- Based Access Control (RBAC): Features and Motivations"

Resumen:Los autores plantean un modelo para controlar los permisos de acceso a cierta información. Este modelo basado en roles tiene varios beneficios con respecto a otros modelos. Los dos principales son:

- Si bien RBAC no promueve ninguna política de protección, ha sido demostrado que soporta principios y políticas de seguridad que son importantes para el área comercial y gubernamental donde se maneja información no clasificada pero sensible.
- Es más sencillo el manejo de permisos, ya que cada usuario está asociado a un rol. Entonces sólo se manejan roles, no usuarios, y eso es bastante menos información.

Discusión: Este es un *paper* fundacional que argumenta fehacientemente las ventajas del modelo de roles que se utilizará en esta memoria.

[5] Alban Gabillon, Léo Letouzey.
"A View Based Access Control Model for SPARQL"

Resumen: Se plantea un modelo de control de acceso basado en vistas para SPARQL, un lenguaje que sirve para expresar consultas en datos guardados como RDF. El propósito del modelo, es lograr una manera de manejar el acceso a los datos. En el *paper* se dan a conocer otros modelos ya presentados para este mismo problema, pero estos no son escalables, es decir, no sirven para grandes volúmenes de datos. Este modelo consiste en asignarle una vista a cada rol, de manera que los usuarios hagan consultas sobre la vista que les corresponde según su rol. Además así los super-usuarios tienen completo control de la información a la cual tiene acceso cada rol. Los problemas que se plantean al momento del manejar de los roles, que aún no han sido resueltos, son las delegaciones, los sub-roles y la negación.

Discusión: El modelo que se presenta en este *paper*, es un modelo que no resuelve el problema de la modificación en la base de datos, ya que a cada rol o usuario se le asigna una vista y no un set de datos. Si bien no resuelve completamente el problema que se quiere resolver, se puede aplicar dentro de la solución. Se podría aplicar un modelo basado en vistas para los usuarios que tienen sólo permiso para consultar, ya que puede darse el caso que el usuario no necesita tener acceso a toda la información. En este caso sería más eficiente que el usuario pudiera hacer consultas sobre una vista.

[6] Sukhoon Lee, Jangwon Kim, Doo-Kwon Baik.
"Two-step Role-Based Access Control method for Ontology Storage".

Resumen: Los autores plantean un modelo de control de acceso para SPARQL, basado en otros dos modelos. Los modelos en los cuales se basa son:

- Reescribir la *query*, es decir, reformular la consulta según los datos a los cuales tiene acceso el rol que está consultando.
- Filtrar resultados de la consulta después de ser ejecutada.

El primero tiene el problema de que no garantiza la privacidad luego de la "inferencia" de las ontologías. El segundo es costoso y tarda mucho tiempo, ya que puede ocurrir que se carguen muchas ontologías que luego serán filtradas, lo que es contraproducente. En el *paper* se discute la posibilidad que combinar ambos métodos. Primero se reescribiría la *query* y luego se filtraría el resultado. Este modelo sería más rápido en la performance de la consulta.

Discusión: Si bien es un modelo que intenta tomar lo mejor de dos modelos, es un modelo costoso. Necesita tanto de un buen parseador de *queries* para reescribirlas y luego un sistema que logre filtrar los resultados de manera eficiente.

[7] Sabrina Kirrane, Ahmed Abdelrahman, Alessandra Mileo, Stefan Decker.
"Secure Manipulation of Linked Data"

Resumen: Los autores demuestran cómo se puede fortalecer DAC sobre un modelo de data RDF con permisos, políticas y Datalog rules.

- Demuestran cómo el jerárquico *Flexible Authorization Framework* puede ser adaptado para trabajar sobre grafos de información.
- Dan una definición formal de una instanciación RDF del *framework*, *Graph based Flexible Authorization Framework or G-FAF*.
- Describen cómo el *pattern matching* y la propagación de reglas pueden ser usadas para mantener las políticas de control de acceso para *linked data*.
- Finalmente, muestran cómo el resolver los conflictos de políticas e integridad pueden asegurar la integridad del control de acceso.

Discusión: El razonamiento sobre reglas de políticas de acceso propuesto, no se incor-

poró en este trabajo porque es para un manejo sofisticado de permisos, y como primera instancia se busca algo más sencillo.

[8] Sabrina Kirrane, Alessandra Mileo, Stefan Decker.
"Applying DAC principles to the RDF graph data model"

Resumen: Al comienzo explica cómo se puede aplicar DAC tanto en el modelo entidad relación como en el modelo de árboles. Luego, expone con más detalle cómo se puede aplicar DAC en RDF. En esta sección parte explicando el modelo RDF, sigue con la asignación de permisos y las reglas de derivación, y termina definiendo las funciones de *'GRANT'* y *'REVOKE'* para SPARQL.

Discusión: Dado que es un modelo donde cada conjunto de datos pertenecen a una persona, y es ella la que puede dar los permisos al resto de los usuarios, se acerca bastante a lo que se quiere como solución al problema que se está planteando. El delegar la administración de los permisos es algo deseable en este proyecto.

[9] Joon S. Park, Ravi Sandhu, Gails-Joon Ahn
"Role-Based Access Control on the Web"

Resumen: Dado que hoy en día el control de acceso en la Web no es escalable a *enterprise-wide systems* porque están mayormente basados en identidades individuales, se plantean 3 diferentes formas para manejar los roles en la Web, para manejar el acceso a los datos. Estos son los siguientes:

- *Cookies* Seguras, que a diferencia de la cookies normales ocupan un método de encriptación.

- Certificados inteligentes, que son únicas para el cliente, es decir, no depende del browser como es el caso de las cookies. Sin embargo, para aplicarlas se necesita ayuda del usuario.

- LDAP(*Ligthweight Diractory Access Control*).

Además se habla de dos tipos de arquitectura *user-pull* y *server-pull*. La ventaja de las arquitectura enfocada en el usuario es que el método de identificación puede ser reutilizable para distintos sistemas. En cambio, en el caso de una arquitectura enfocada en el servidor, la ventaja es que es mucho más sencillo para sistemas donde los roles son dinámicos. Las cookies son las únicas que no pueden aplicarse en una arquitectura *server-pull*, porque son guardadas en la máquina del usuario.

Discusión:No se utilizará LPAP en este trabajo, pues limita bastante al momento de aplicarlo.

3.3. Proyectos Relacionados

A continuación se presentan los proyectos desarrollados que más relación tenían con esta memoria. Por un lado se encuentra el *endpoint* Virtuoso ya tiene solucionado el tema de seguridad. Como se menciona en la sección 3.1.2 Virtuoso utiliza como base el modelo de control de acceso que poseen las bases de datos hoy en día. Si bien el sistema de control de acceso implementado en Virtuoso es bastante similar a lo que se quiere realizar en este proyecto, la modularidad es uno de los principales objetivos de este proyecto y la manera en se maneja en Virtuoso no lo es. El sistema fue diseñado exclusivamente para ese *endpoint*, es decir, no es lo que se busca.

Por otro lado, se encuentra el sistema Shi3ld [3] que se plantea de forma muy similar a este proyecto, es decir, un filtro para *endpoints* SPARQL genérico, que no necesita intervenirlos al momento de la integración. Luego de leer sobre él, se contactó a Luca Costabello[1], uno de los desarrolladores de Shi3ld, para saber un poco más del desarrollo de Shi3ld. Actualmente ninguno de los desarrolladores del equipo continúa el proyecto, por lo que se consideró la posibilidad de tomar el código *open-source* como base, en vez de programar todo desde cero. De la información entregada por el desarrollador mencionado anteriormente, se pudo extraer cuáles eran las falencias y/o diferencias con respecto a lo que busca este proyecto.

Shi3ld, consta de dos partes, una interfaz gráfica para el manejo de las políticas y un *Web service* que aplica las políticas y se comunica con el *endpoint*. Dentro del trabajo futuro de Shi3ld consideraron las siguientes funcionalidades, por ejemplo:

- Administración de usuarios, por ahora sólo hay un usuario, administrador de las políticas.
- Diferente interfaz gráfica según el rol que tiene el usuario. Hoy en día no existe interfaz gráfica que permita a un usuario hacer consultas, donde las políticas estén siendo aplicadas.

Para manejar el acceso sobre la información, Shi3ld implementó un sistema de políticas basadas en contexto, es decir, dado el contexto del usuario, Shi3ld decide si le da permiso o no a un recurso. Por ejemplo, hay políticas que aplican para usuarios que se encuentran dentro de una cierta área geográfica. Cada una de estas políticas está asociada a un *Named Graph*[2], que básicamente es un grafo más con identificadores.

Para conocer un poco más allá el desarrollo de Shi3ld, se buscó instalarlo y empezar a probar las distintas funcionalidades. La documentación de esto era insuficiente, por lo que el Sr. Costabello, colaboró en el proceso. Al ir un poco más allá en el código, se notó que el *Web service* no permite hacer *queries*, algo relevante dentro de este proyecto. Shi3ld permite acceder a los recursos, que en este caso serían los *Named Graphs*.

[1] `http://luca.costabello.info/` Accesado 6 Abril 2015

[2] `http://wimmics.inria.fr/projects/shi3ld/` Accesado 1 September 2014

Este producto carece de varias funcionalidades que este proyecto busca desarrollar y el enfoque de seguridad es distinto. Dado esto y el hecho de que entrar en código ajeno no es algo simple, se decidió partir desde cero el desarrollo. Sólo se utilizó la función que realiza los accesos al *endpoint*.

Capítulo 4

Solución

4.1. Modelo Solución

La solución al problema, como se mencionó anteriormente, es modular, de manera que se pueda incluir en todos los *endpoints*, que reciban los *requests* mediante HTTP, sin tener que hacer cambios mayores. Si se tiene un módulo que maneja los accesos, el que quiera crear un nuevo *engine* SPARQL, no tendrá que preocuparse de la capa de permisos porque podrá ponerlo detrás de este *frontend*. Además varios *engines* SPARQL podrán ser accedidos bajo el mismo *frontend*, con los mismos usuarios y roles.

Si bien puede ser que el usuario que expuso el grafo en la Web, no sea el dueño de esa información, en esta solución se aplicó el modelo de control de acceso DAC. Este modelo como se explicó anteriormente, plantea que el que expone la información, es el que maneja el acceso sobre ella. Se decidió hacer esto, pues es el que fue requerido cuando se comenzó el proyecto. Una extensión natural es tener un administrador común, funcionalidad que necesita evaluarse (necesidad, factibilidad, etc).

La Figura 4.1 muestra cómo será la jerarquía de usuarios. En la cima tenemos a un administrador, quien será el que dará permiso para que un usuario pueda exponer su información. Luego tenemos a los dueños de los grafos, quienes manejan los permisos sobre sus grafos. Por último están los usuarios que sólo pueden hacer modificaciones en los grafos a los cuales tienen permisos.

La Figura 4.2 presenta el modelo de la solución. Comienza con una interfaz muy similar a la que tienen hoy en día todos los *endpoints*, donde el usuario puede escribir consultas en lenguaje SPARQL. La idea es que el usuario común que sólo busca consultar información pueda hacerlo igual que lo hace hoy, es decir, sin tener que registrarse en la página. Para el resto de los usuarios que no busca sólo consultar, si no que busca hacer modificaciones en la información o administrar permisos sobre sus grafos debe hacer *Log In*. Al momento

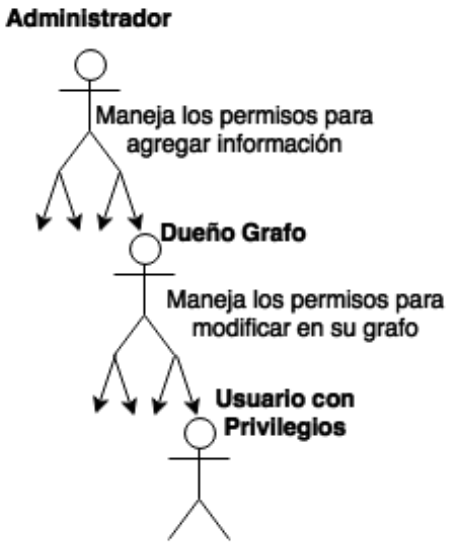

Figura 4.1: Estructura de los usuarios con permisos, es decir, usuarios que están registrados.

de registrarse pueden darse tres escenarios: que el usuario sea un usuario con permisos de modificación sobre un grafo o que el usuario sea dueño de un grafo o que sea ambos, es decir, dueño de un grafo y con permisos para modificar un grafo distinto al suyo. En el caso de ser el dueño de un grafo podrá mediante una interfaz gráfica manejar los permisos.

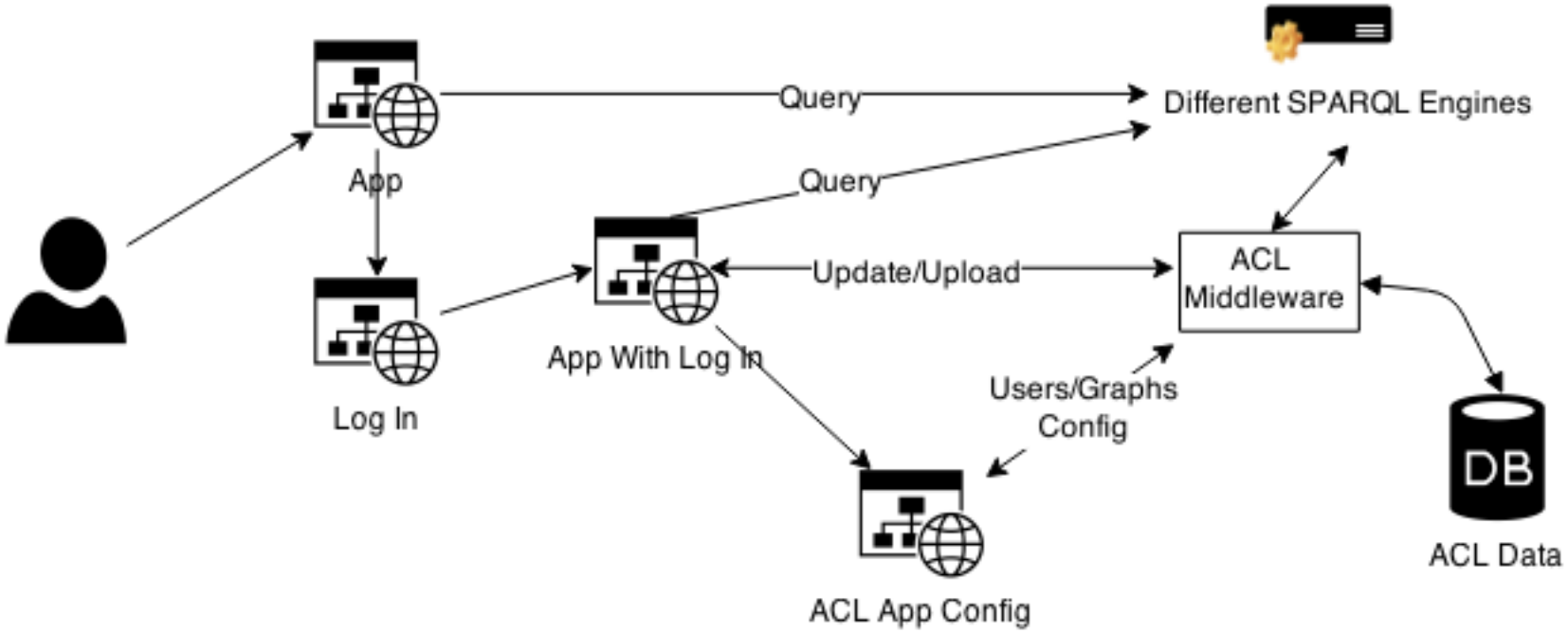

Figura 4.2: Diagrama Solución.

El diagrama de la solución se puede dividir en tres partes:

- *Endpoint*
- Interfaz gráfica
- ACL Middleware

A continuación se detallará cada una de ellas.

4.1.1. *Endpoint*

El *endpoint* utilizado en este proyecto fue Fuseki, ya que es open-source y ha sido utilizado por el implementador. Además Fuseki, a diferencia de Virtuoso, no tiene un sistema de seguridad. Los accesos al *endpoint* son mediante llamadas HTTP, lo que permite que el diseño sea flexible y no quede restringido a un solo *endpoint*.

4.1.2. Interfaz Gráfica

La interfaz gráfica dentro de este proyecto no maneja la lógica del sistema de seguridad, sino que es un intermediario entre el usuario y la API. Las funcionalidades disponibles dentro de la página Web dependen del rol del usuario. A continuación se detallará qué es lo que está disponible para cada rol.

Acción	**Administrador**	**Dueño del Grafo**	**Usuario con Privilegios**
Administración usuarios con rol "Dueño Grafo"	X		
Subir grafos	X	X	
Update de sus grafos o grafos que tiene asignados	X	X	X
Administración de permisos de sus grafos.	X	X	
Creación de usuarios con rol "usuario con privilegios"	X	X	

Tabla 4.1: Tabla de Permisos

Dado que dentro del diseño se definió que todo usuario podía consultar el endpoint, las consultas van directo, es decir, no pasan por el middleware.

4.1.3. *ACL Middleware*

Sesión

Dado que uno de los objetivos de esta API, es que pueda ser utilizada desde cualquier interfaz gráfica, es decir, no sólo la que se hizo durante este proyecto, se decidió que la sesión

la manejará la misma API. Toda petición que se haga a la API, excepto el *Log In*, debe incluir el ID de sesión para verificar la identidad del usuario.

Datos ACL

Un *endpoint* que incluya esta plataforma de seguridad será utilizado para almacenar grafos de gran volumen con información que necesite protección contra usuarios mal intencionados. Dado lo anterior se sigue que, la cantidad de usuarios registrados y la cantidad de grafos por *endpoint* será reducida. Por lo tanto, la información de las sesiones, usuarios y asignación de grafos será guardada en archivos xml de configuración. Sin embargo, el acceso a cada uno de estos archivos está bien modularizado, de manera que si se llega a la conclusión que guardar esta información en archivos no es la mejor manera, será simple cambiar los archivos a un modelo bases de datos.

Métodos

La API debe incluir métodos que permitan realizar las siguientes acciones

- *Log In*
- Agregar un nuevo usuario, con distintos roles
- Asignarle/Quitarle permisos de modificación sobre un grafo a un usuario.
- Decidir si el usuario tiene o no permisos para hacer un *update*
- Agregar/Eliminar un grafo
- Obtener el rol del usuario según su sesión
- Obtener los grafos que un usuario puede modificar

Parseo *Updates*

La funcionalidad más importante del *Middleware* es la que parsea el *update* y dice si el usuario tiene o no permisos para hacerlo.
A continuación se mostrará un ejemplo de cada una de las sintaxis posibles de un *update* en SPARQL .

Ejemplo Nº1:
PREFIX a: <http://example.org/>
INSERT DATA{GRAPH a:ng1
{ <http://example.org/book/books5>
<http://purl.org/dc/elements/1.1/creator>
”J.K. Rowling”} }

Ejemplo Nº2:
DELETE DATA{GRAPH <http://example.org/ng1>

{ <http://example.org/book/books5>
<http://purl.org/dc/elements/1.1/creator>
”J.K. Rowling”} }

Ejemplo Nº3:
INSERT { GRAPH <http://example.org/ng3>{ ?o ?p ?q} }
USING <http://example.org/ng1>
WHERE { ?o ?p ?q }

Ejemplo Nº4:
WITH <http://example.org/ng3>
DELETE { ?o ?p ?q }
WHERE { ?o ?p ?q }

El parse del *update* tiene que lograr identificar a cuáles son los grafos que el usuarios está buscando modificar según la URI.

4.2. Implementación

La implementación de la solución se dividió en dos partes, tal cual se ve representado en el diagrama de la arquitectura. Por un lado está la página Web con la que interactuará el usuario y, por otro lado, el servicio REST, que actuará como *middleware* entre el *endpoint* y la interfaz gráfica.

4.2.1. Interfaz Gráfica

La interfaz gráfica de la aplicación será implementada en PHP, pero en realidad no es muy relevante el lenguaje en sea desarrollada porque la lógica la maneja la API. Es similar a la que hay actualmente en cada *endpoint*, adicionando la posibilidad de hacer *Log In* y una interfaz de configuración para los usuarios con más privilegios. A continuación se mostrarán las diferentes pantallas.

La Figura 4.3 es la página principal de la aplicación permite a los usuarios no registrados consultar la información expuesta en el endpoint que está asociado a la página. Además permite al usuario registrado hacer *Log In*.

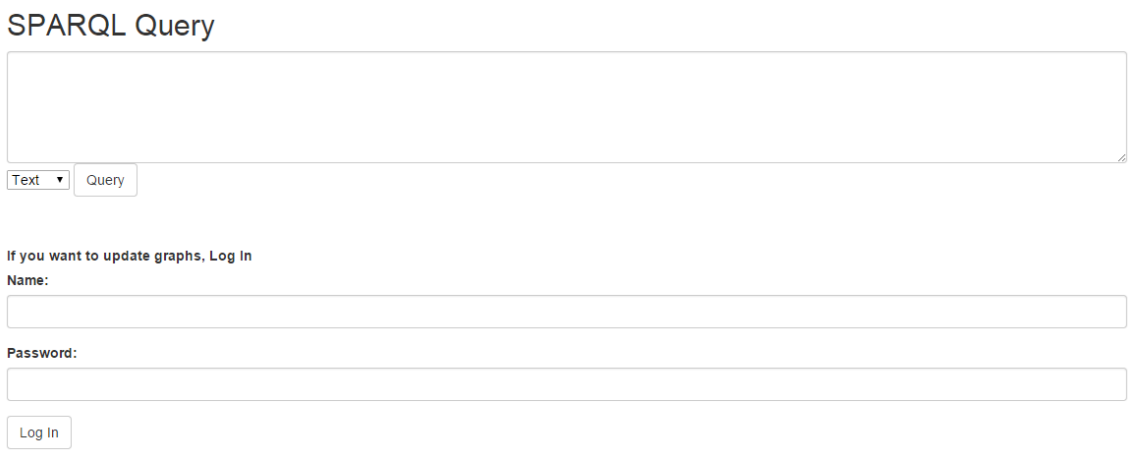

Figura 4.3: Pantalla principal.

La Figura 4.12 es lo primero que ve un usuario luego de hacer *Log In*. Dentro de esta pantalla hay tres cosas que se pueden hacer. Primero se puede hacer consultas, al igual que en la pantalla anterior. Segundo, se pueden hacer *updates* sobre los grafos que el usuario tiene asignados. Finalmente, si el usuario tiene el rol de "Dueño del Grafo" puede subir un nuevo grafo.

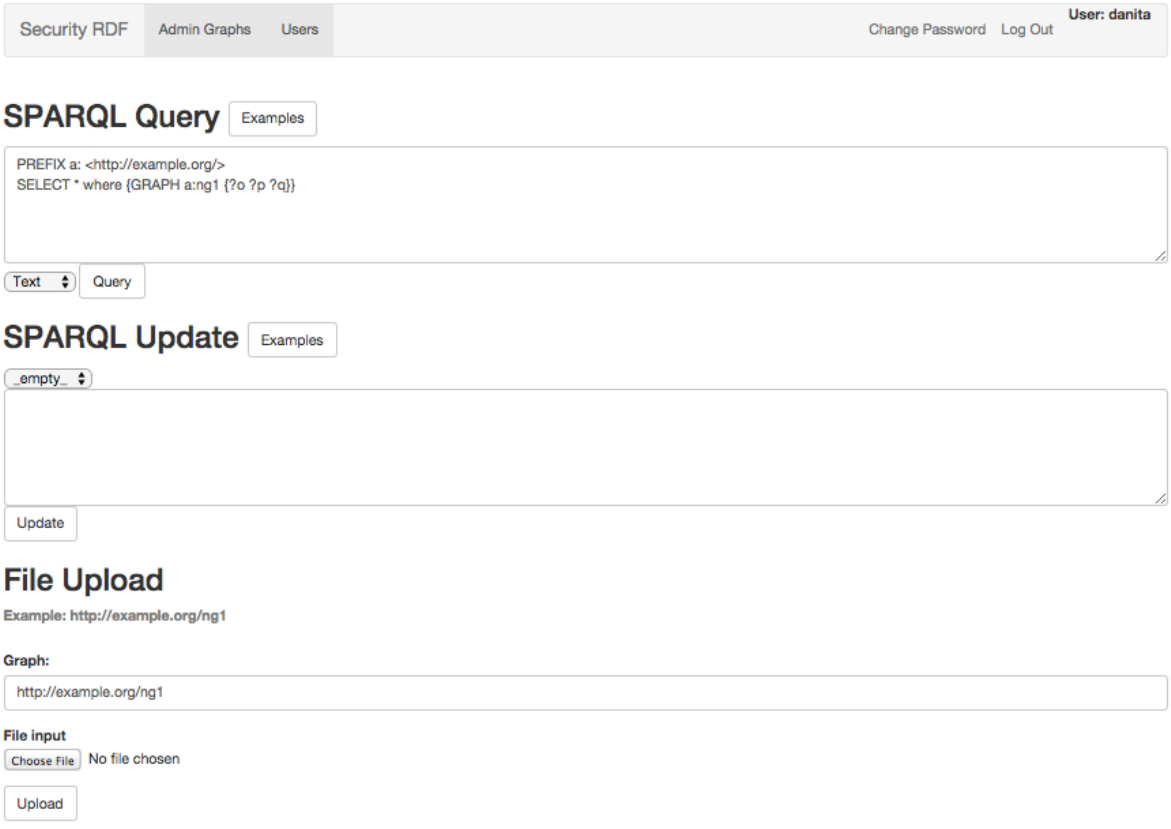

Figura 4.4: Pantalla donde se pueden realizar los *updates*.

La Figura 4.5 es una página que pueden ver los usuarios con rol de "Administrador"o de

"Dueño del Grafo". En ella se administran los grafos y los usuarios que tienen permisos de modificación sobre esos grafos. Además se pueden agregar usuarios.

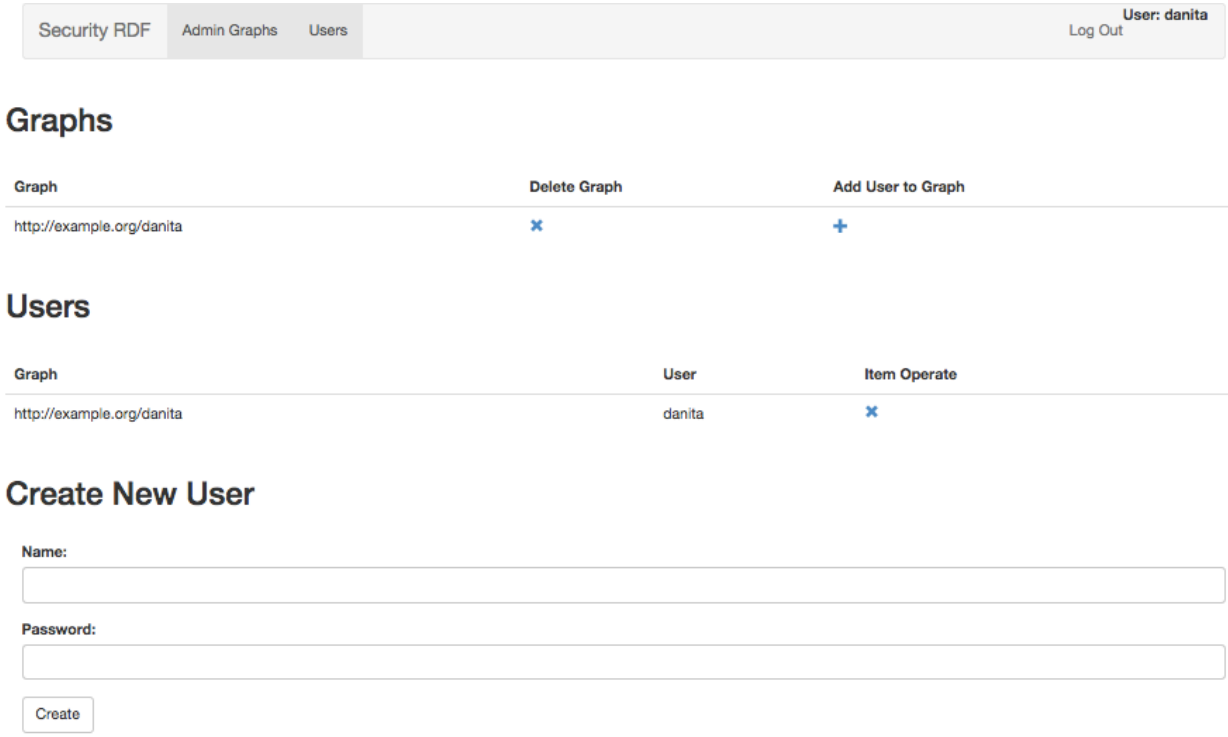

Figura 4.5: Pantalla de administración de grafos.

La Figura 4.6 es una página que pueden ver los usuarios con rol de "Administrador". En ella se administran los usuarios que tendrán el rol de "Dueño del Grafo".

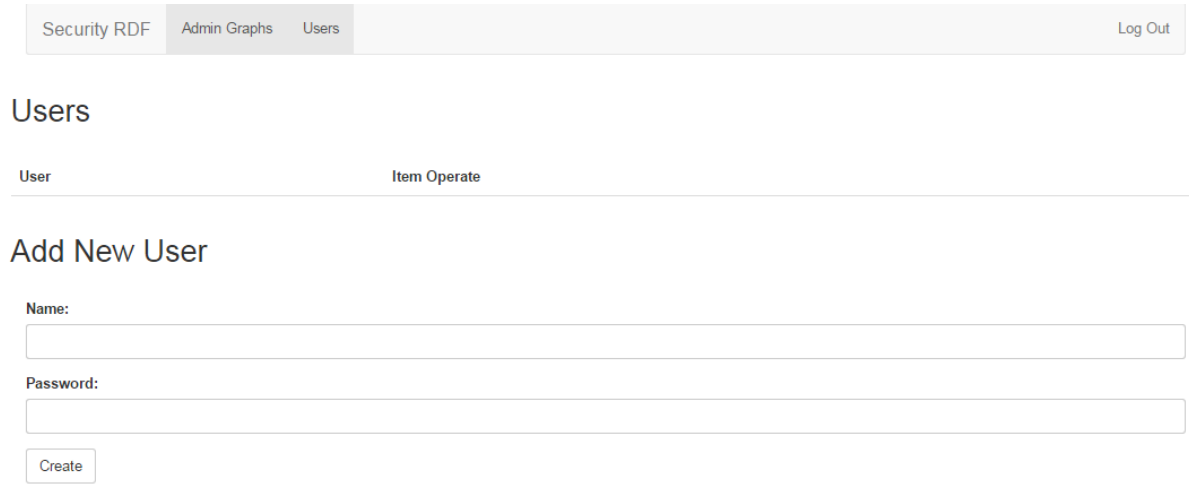

Figura 4.6: Pantalla de administración de usuarios que tienen permisos para agregar grafos.

4.2.2. *ACL Middleware*

El módulo de *ACL Middleware* es un servicio REST implementado en JAVA con el *framework* Spring. Se escogió JAVA, porque es un lenguaje estable y portable. Además el *endpoint* sobre el cual se trabajará, fuseki, y Shi3ld están desarrollados en JAVA. Se escogió Spring como *framework* porque es lo más utilizado para desarrollo Web en JAVA, y ya ha sido utilizado por el implementador.

Archivos de Configuración

Sesiones

El archivo sessions.xml mantiene las sesiones vigentes. El xml tiene la siguiente estructura:

```
<properties>
<comment/>
<entry key="4bc04d505f652f30c5abb71b2a76cc0e.time">1428003549203</entry>
<entry key="4bc04d505f652f30c5abb71b2a76cc0e.ip">200.86.0.53</entry>
<entry key="4bc04d505f652f30c5abb71b2a76cc0e">dani</entry>
</properties>
```

Por cada uno de las sesiones tiene el nombre del usuario y el *timestamp* en que fue ingresada.

Usuarios

El archivo users.xml guarda los usuarios con sus respectivas claves y roles. El xml tiene la siguiente estructura:

```
<properties>
<comment/>
<entry key="dani.graphs">http://exampleTest.com/ng1;http://exampleTest.com/ng2</entry>
<entry key="dani.pass">81dc9bdb52d04dc20036dbd8313ed055</entry>
<entry key="dani.role">admin</entry>
</properties>
```

Grafos

El archivo graphs.xml guarda los grafos con los usuarios que tienen permisos sobre ellos. El xml tiene la siguiente estructura:

```
<properties>
<comment/>
<entry key="http://exampleTest.com/ng1__admin">coni;vale;</entry>
</properties>
```

Por cada grafo hay una lista de usuarios que tienen permiso para hacer modificaciones en el grafo.

Métodos API REST

- **logIn**: Método que retorna un ID de sesión si el usuario está dentro de los usuarios válidos y si la clave es la que corresponde.
- **addUser**: Método que permite agregar un usuario, pero sin antes chequear que el que está haciendo la petición sea un usuario con esos permisos.
- **addUserToGraph**: Método que permite al dueño de un grafo asignarle a un usuario permisos de modificación sobre ese grafo.
- **deleteUserFromGraph**: Método que permite quitarle los permisos de modificación a un usuario sobre cierto grafo.
- **deleteGraph**: Método que permite eliminar un grafo del *endpoint*.
- **getUserRole**: Método que permite obtener el rol asignado al usuario.
- **getUserGraphs**: Método que permite obtener los grafos del usuario.
- **upload**: Método que permite agregar un grafo a partir de un archivo.

Parseo *Updates*

En la sección 4.1.3 se dieron a conocer las distintas sintaxis que existen en SPARQL para hacer *updates*. Al momento de parsear un *update* no sólo basta ver las URIs que están presentes en él, ya que un usuario dentro de un *update* puede estar modificando un grafo A insertando datos de un grafo B. En ese caso el usuario tiene que tener permisos sólo sobre el grafo A. Dado esto se analizó las distintas sintaxis y se encontró un patrón. Dentro de un *update*, los grafos que van a sufrir modificaciones son las URIs que son precedidas palabra "GRAPH". Ya con el partrón identificado, el parseo no fue complejo. Ya teniendo el patrón identificado, fue más sencillo identificar los grafos que el usuario buscaba modificar.

El lenguaje SPARQL está en constante evolución, ya que busca tener las mismas funcionalidades que el lenguaje SQL. Si dentro de esta evolución cambia el patrón encontrado dentro este proyecto, la función que parsea tendrá que ser cambiada.

4.2.3. Documentación

El código de este proyecto quedará abierto, es decir, podrá ser utilizado por cualquiera que quiera agregar funcionalidades o modificarlo. Tanto el código como la documentación están publicados en *Github*. La interfaz gráfica junto a su documentación se encuentran en el siguiente repositorio `https://github.com/danif88/Endpoint-Security-Interface.git`. En el caso de la API se encuentra en `https://github.com/danif88/Endpoint-Security.git`.

El código de la aplicación quedó con licencia Atribución 4.0 Internacional de Creative Commons.

4.3. Tests Unitarios

Todo proyecto de ingeniería que sufrirá grandes modificaciones en el tiempo debe incluir su conjunto de tests unitarios para asegurarse que al momento de agregarle una nueva funcionalidad las anteriores sigan funcionando correctamente. Este proyecto está destinado a sufrir modificaciones, ya que las bases de datos RDF son algo que aún se está investigando, y sin ir más lejos el lenguaje SPARQL va a seguir evolucionando, ya que aún existen acciones que se pueden hacer en bases de datos comunes que no se pueden hacer en SPARQL.

Hoy en día los dos *frameworks* más populares para tests unitarios en Java son: JUnit[1] y TestNG[2]. Ambos son bastante similares y te permiten testear el código de manera rápida y efectiva. La gran diferencia entre ambos es que TestNG te da la opción de *Parametrized Testing*, que al momento de correr tests de gran envergadura es muy útil. Sin embargo, en un proyecto como este en que los test que se tienen que realizar no tienen una gran complejidad, el framework TestNG le queda grande, se gasta más tiempo configurando que construyendo los tests.

Los tests unitarios que se realizaron se hicieron con el objetivo de testear cada uno de los métodos del servicio REST. Durante su construcción se buscó una herramienta para simular las llamadas REST. Spring incluye un módulo específico para esto que se llama MockMvc[3].

Gracias a estas dos herramientas se construyeron los tests exitosamente. Se realizó un test por cada caso de cada método de la API.

[1] `http://junit.org/` Accesado 6 Abril 2015

[2] `http://testng.org/doc/index.html` Accesado 6 Abril 2015

[3] `http://docs.spring.io/spring-framework/docs/3.2.0.RC2/api/orgspringframework/test/web/servlet/MockMvc.html` Accesado 6 Abril 2015

4.4. Usabilidad

La interfaz gráfica del proyecto es bastante similar a la que tiene Fuseki en estos momentos, incluyendo las nuevas funcionalidades. La usabilidad de estas nuevas funcionalidades integradas a la interfaz no ha sido testeada. Por esta razón, es necesario hacer un estudio de usabilidad antes de poder decir que el producto está listo.

Los conocimientos en estudios de usabilidad de la memorista son los obtenidos en el Taller de Usabilidad, dictado por el profesor Jaime Sánchez. A partir de estos, se definió como objetivo el evaluar los siguientes atributos:

- Errores
- Satisfacción

Para cumplir con el objetivo, durante el estudio se aplicarán dos métodos, uno cuantitativo y otro cualitativo. Primero se aplicará el método de Observación con Thinking Aloud, donde el atributo evaluado será principalmente errores. Luego se aplicará el Cuestionario de Usuario Final, un instrumento cuantitativo para valorar la satisfacción de los usuarios.

4.4.1. Muestras

Las personas que están dentro de la muestra son personas que ya tienen experiencia con bases de datos RDF.

Observación con Thinking Aloud:

- 4 personas

Cuestionario de Usuario Final

- 10 personas

4.4.2. Instrumentos

Observación con Thinking Aloud:

- Block de notas
- Pauta de Observación (Anexo)

Cuestionario de Usuario Final

- Se tomó como base un cuestionario construido por el experto en usabilidad para aplicaciones Web, el profesor Jaime Sánchez. (Anexo)

4.4.3. Procedimiento

Cuestionario Usuario Final:

Para aplicar este cuestionario se tuvo que pensar la manera en que se asignarían los usuarios a cada uno de los encuestados. Primero se pensó en un usuario común , pero eso significaba que los ellos no podrían conectarse al mismo tiempo. Lo que se hizo finalmente, fue crear una lista de 30 usuarios en un *Google Doc*, para que cada encentado pudiera marcar cual de los usuarios estaba utilizando.

El cuestionario se creó en la página *e-encuesta*, para que al enviarlo fuera sólo un *link* y para que al obtener los resultados fuera sólo descargar una planilla excel.

Observación con Thinking Aloud:

Para aplicar este método se procedió de la siguiente manera. Se hizo la observación a cada uno de los observados por separado. Además, luego de cada observación se hizo mejoras en el sistema según las falencias que se habían encontrado. De esta forma la observación, los resultados fueron mejorando a medida que avanzaban los observados.

4.4.4. Resultados

Resultados Cuestionario Usuario Final:

Los resultados obtenidos al aplicar el cuestionario se encuentran detallados en el Anexo 3. A continuación se muestra un gráfico con las preguntas más relevantes del cuestionario. Dentro del gráfico no se incluyeron las preguntas relacionadas con imágenes y colores porque en este caso no hay imágenes dentro del sitio.

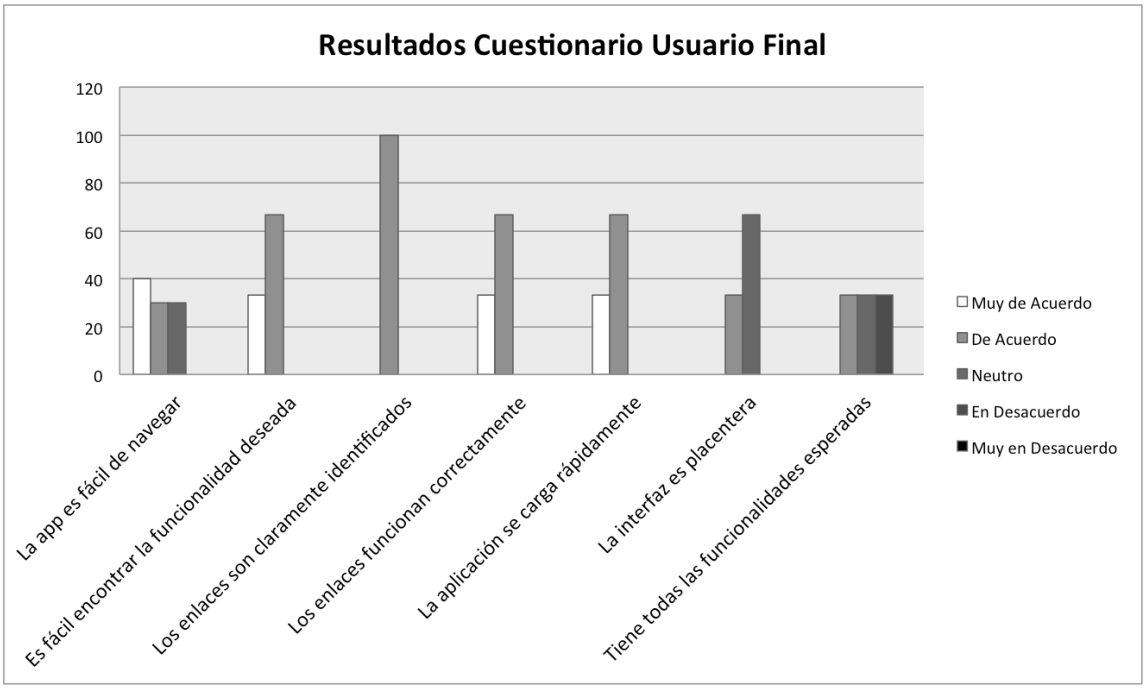

Figura 4.7: Resultados Cuestionario Usuario Final.

En el gráfico se puede observar que los resultados fueron positivos, en ninguno de las preguntas tuvimos como resultado "Muy en Desacuerdo".

Resultados Observación con Thinking Aloud:

En esta sección presentaremos los resultados de cada uno de los evaluados. Por cada uno de los evaluados se completó una pauta de observación. La pauta de cada uno de los evaluados se encuentra en el Anexo 4. Lo que se presentará a continuación es el análisis de esos resultados.

Primer Observado: C.G.1

C.G.1 cumplió todas las tareas con éxito pero tuvo dificultades al momento de hacer las consultas y los *updates* porque no se acordaba de la sintaxis. Él sugirió que se agregaran ejemplos, de manera que el usuario pudiera copiarlas. A continuación se encuentra la pantalla luego de la modificación. Tanto para las consultas como para los *updates* se agregó un botón que levanta un *pop up* con los ejemplos.

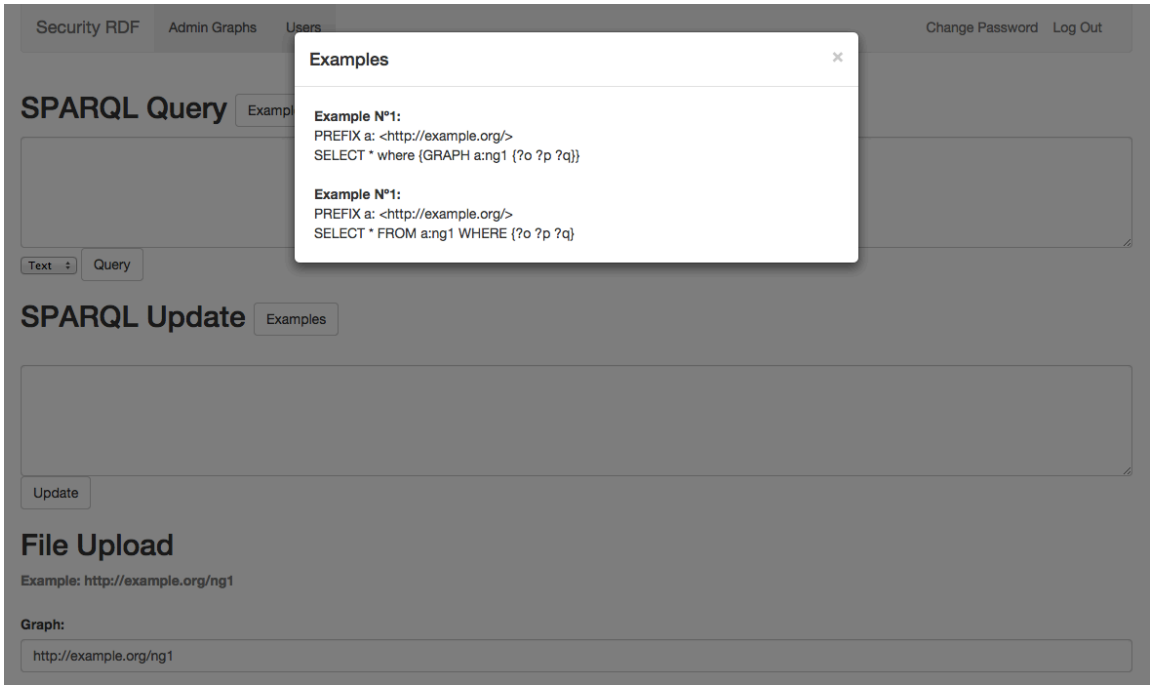

Figura 4.8: Pantalla donde se pueden realizar los *updates*, versión 2.

Segundo Observado: C.G.2

C.G.2 cumplió todas las tareas con éxito, pero sugirió que en algunos *endpoints* viniera una consulta básica ya escrita en el *input*. Esta sugerencia era razonable, por lo tanto, ahora se incluye la consulta básica dentro del *input* donde se realizan las consultas.

Tercer Observado: M.Q.

M.Q. tuvo dificultades al momento de asignarle los grafos a un nuevo usuario. Primero comentó que en vez de "*Add New User*" debiera ser "*Create New User*", porque el sintió que estaba agregando a un usuario al grafo en vez de crearlo. La primera versión que se hizo de esta pantalla es la siguiente:

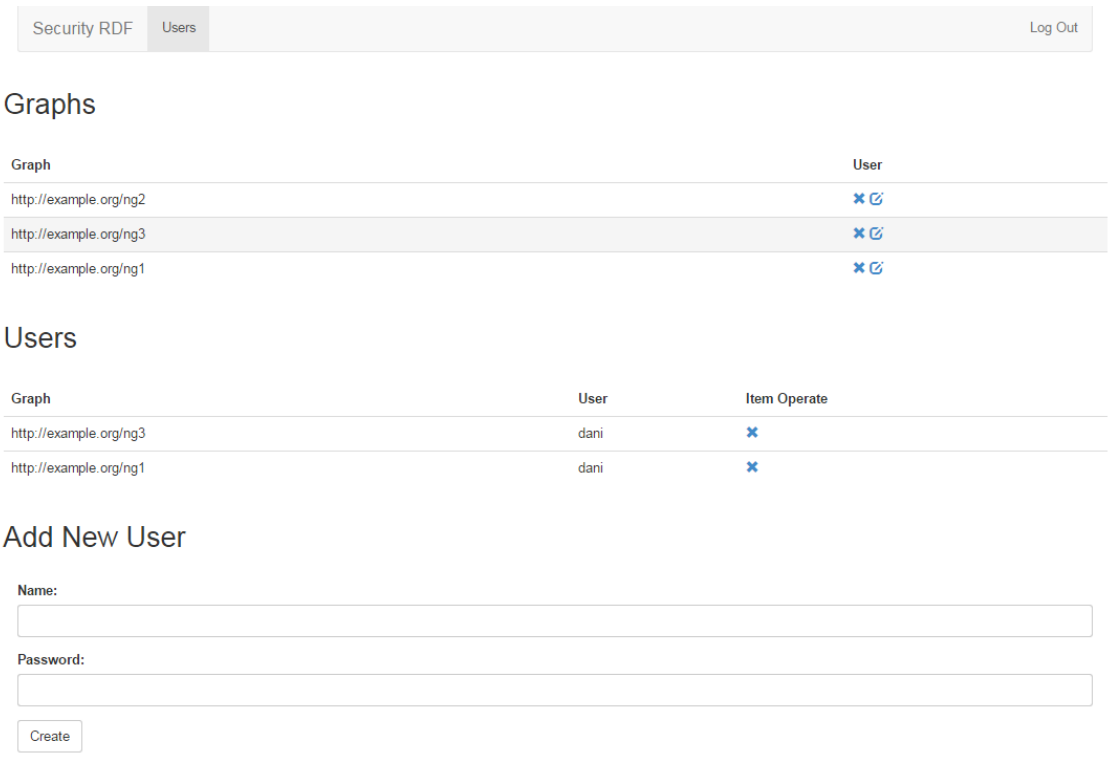

Figura 4.9: Pantalla de administración de grafos, versión 1.

Luego de las observaciones quedó como sigue:

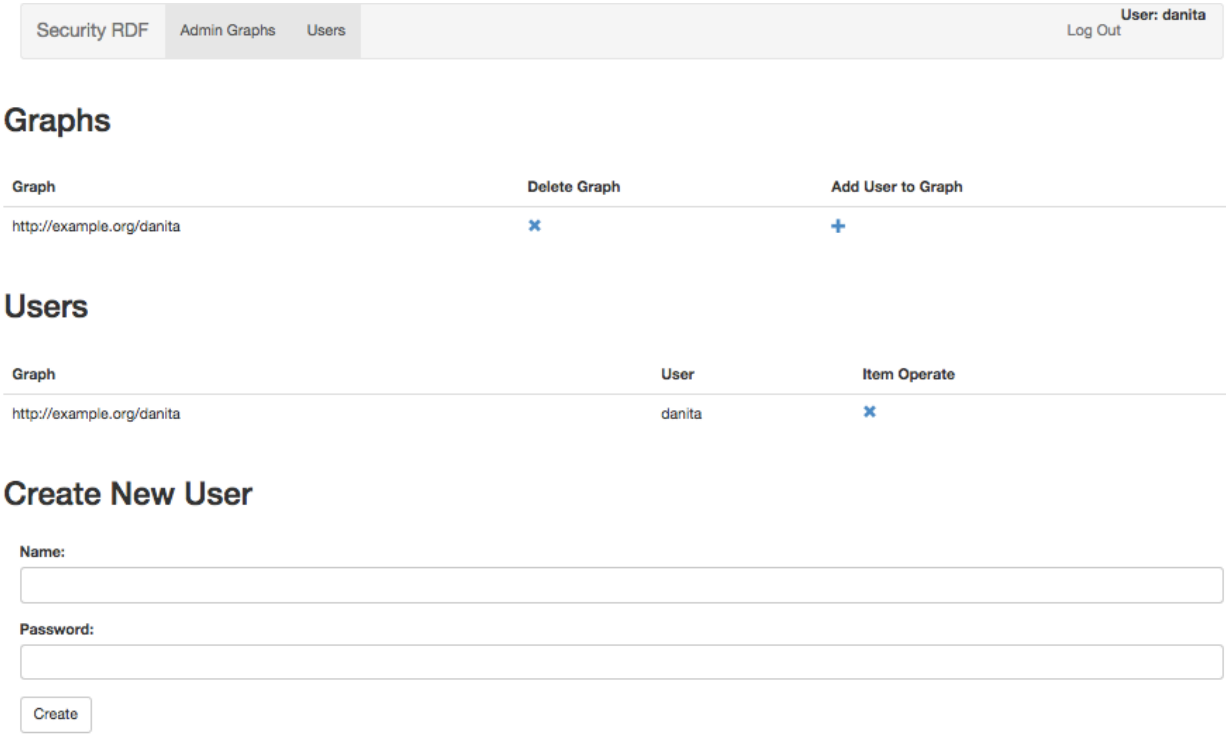

Figura 4.10: Pantalla de administración de grafos, versión 2.

Cuarto Observado: D.H.

D.H. comentó que sería bueno saber el nombre del usuario que está conectado, por lo que se hizo el siguiente cambio en el *header* de la página:

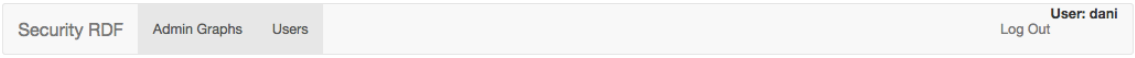

Figura 4.11: *Header*, versión 2.

Además comentó que al momento de asignar un usuario a un grafo se pudieran ver los usuarios que ya están en el sistema. Por lo tanto, ahora en vez de ser un *input* es un *combobox*.

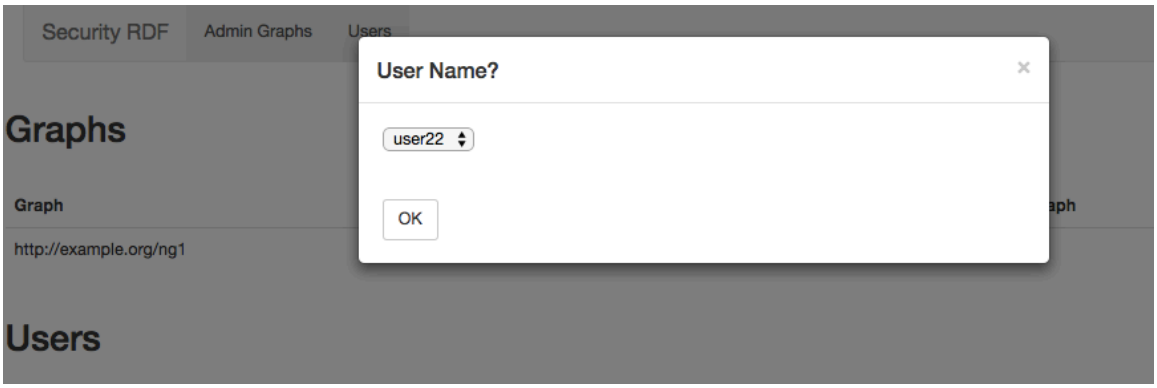

Figura 4.12: Pantalla para agregar un nuevo usuario a un grafo, versión 2.

4.5. Pruebas de Estrés

Al momento de realizar las pruebas de estrés sobre el sistema, se utilizó la herramienta JMeter[4], que permite realizar pruebas de *performance* sobre recursos tanto estáticos como dinámicos. Y soporta HTTP, protocolo utilizado en este proyecto.

El objetivo de las pruebas de estrés es evaluar cuánto afecta este sistema de seguridad al tiempo de respuesta, es decir, si el tiempo de respuesta se ve muy afectado al momento de hacer un update. Un *endpoint* recibe más *requests* que consultan la data expuesta que *resuests* que modifican esta data. Por lo tanto, los usuarios que hacen *Log In* no son muchos, son más que nada los dueños de los grafos. Cada *endpoint* tiene pocos grafos pero grandes, entonces se supuso que la cantidad de usuarios que tuvieran acceso a estos grafos fluctuarían entre los 10 - 50. Después de definir el objetivo se decidió que los tests para cada modalidad (Update directo a fuseki, Update pasando por el *middleware*) serían los siguientes:

- 10 usuarios insertando 100 registros

[4] `http://jmeter.apache.org/` Accesado 6 Abril 2015

- 10 usuarios insertando 1000 registros
- 10 usuarios insertando 10000 registros
- 50 usuarios insertando 100 registros
- 50 usuarios insertando 1000 registros
- 50 usuarios insertando 10000 registros

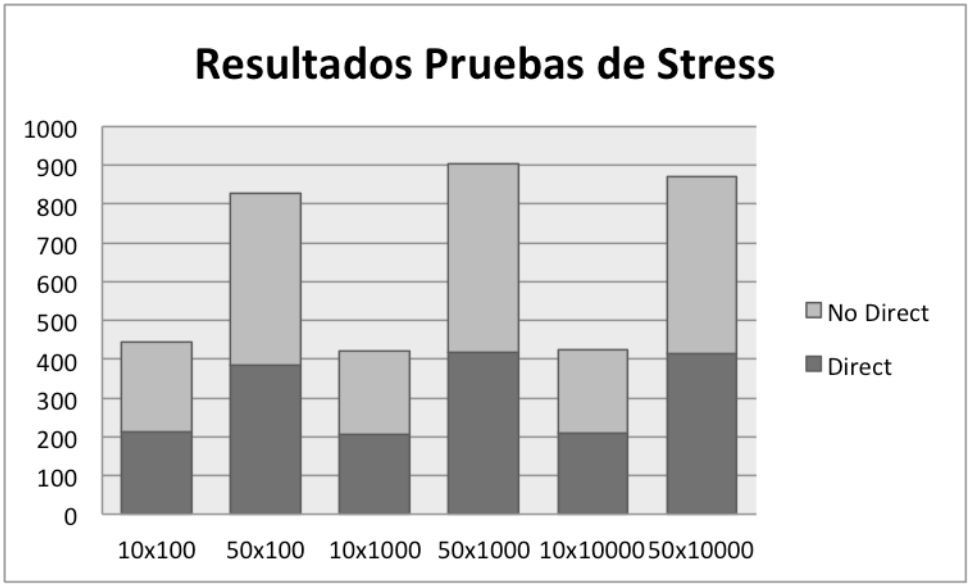

Figura 4.13: Gráfico donde se refleja la diferencia entre hacer *updates* usando el *middleware* y hacerlos directamente en el *endpoint*.

El gráfico anterior refleja que el ponerle seguridad a un *endpoint* no afecta de manera notable el *performance* de los *updates*.

La siguiente prueba de stress que se hizo fue para ver si el tiempo de inserción aumentaba mucho con la cantidad de usuarios. Los resultados fueron los siguientes:

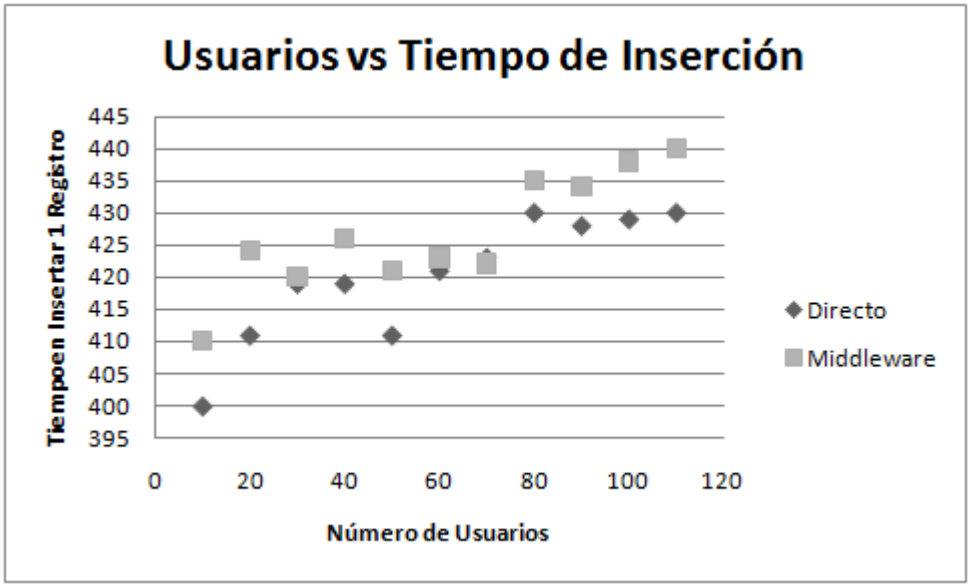

Figura 4.14: Gráfico donde se refleja el tiempo de inserción a medida que aumenta la cantidad de usuarios.

El gráfico anterior refleja que la cantidad de usuarios es directamente proporcional al tiempo de inserción. Si bien el tiempo de inserción aumenta con la cantidad de usuarios, el hecho de hacerlo a través del middleware no afecta de manera

Capítulo 5

Conclusión

El objetivo de este proyecto como se mencionó en la sección 1.2 era resolver el problema de seguridad que tienen hoy en día los *endpoints*. Para cumplir con el objetivo se partió con una investigación, que no sólo fue leer si no que se tomó contacto con investigadores del extranjero que estaban o habían estado desarrollando en esta misma área. Luego, se diseñó una solución, siguiendo algunos conceptos que se encontraron en los *papers* leídos. Por último se implementó la solución. Como toda aplicación tuvo que pasar por un proceso de validación. Esta validación incluyó una evaluación de usabilidad y pruebas de estrés.

El proyecto cumplió con sus objetivos. Primero es una aplicación que permite controlar el acceso a los datos de un *endpoint*, es decir, si a un usuario no le han asignado un grafo no va a poder modificarlo. Además la aplicación demostró que reaccionaría bien cuando estuvieran varios usuarios haciendo uso de ella. Segundo, es genérica, ya que los accesos al *endpoint* se hacen mediante llamadas HTTP. Tercero, pero no menos importante, se logró una aplicación con una interfaz placentera, donde los usuarios logran encontrar las funcionalidades deseadas.

Finalmente, durante el desarrollo y el período de validación salieron propuestas interesantes. Una primera propuesta, es exponer un conjunto de prefijos comunes que estén presente en los datos para facilitar las consultas. Otra propuesta, es el poder administrar más de un *endpoint* con una misma interfaz.

Anexos

Anexo 1: Cuestionario para Usuario Final

El cuestionario que se encuentra a continuación es el que fue aplicado para evaluar la usabilidad del sistema.

Nombre			
Sexo	Masculino		Femenino
Edad			
Experiencia en Endpoints	Pocos Minutos	Horas	Días

Pregunta	**Muy de Acuerdo**	**De Acuerdo**	**Neutro**	**En Desacuerdo**	**Muy en Desacuerdo**
La app es fácil de navegar					
Es fácil encontrar la funcionalidad deseada					
Los enlaces son claramente identificados					
Los enlaces funcionan correctamente					
La aplicación se carga rápidamente (≤30 segundos)					
El uso de las imágenes es aceptable					
El uso del color es aceptable					
El diseño general es apropiado					
La interfaz es placentera					
Tiene todas las funcionalidades esperadas					

Pregunta	**Excelente**	**Bueno**	**Neutro**	**Regular**	**Deficiente**
¿Cómo califica globalmente la app?					

Anexo 2: Pauta de Observación

A continuación se encuentra la pauta que se utilizó al momento de aplicar del método de "Observación con Thinkig Aloud".

Nombre			
Sexo	Masculino		Femenino
Experiencia en Endpoints	Pocos Minutos	Horas	Días

Observación o Tarea	**Logrado Fácilmente**	**Logrado con Dificultad**	**No Logrado**	**Comentarios**
Realizar Query				
Log In				
Log In(super user)				
Subir Grafo				
Update en Grafo (grafo permitido)				
Update en Grafo (grafo no permitido)				
Agregar Usuario				
Agregar Usuario a Grafo				
Cambiar Clave				
Log Out				

Anexo 4: Resultados Cuestionario para Usuario Final

La tabla que sigue refleja los resultados obtenidos luego de aplicar el Cuestionario de Usuario final a 10 usuarios.

Pregunta	Muy de Acuerdo	De Acuerdo	Neutro	En Desacuerdo	Muy en Desacuerdo
La app es fácil de navegar	40 %	30 %	30 %		
Es fácil encontrar la funcionalidad deseada	33.3 %	66.7 %			
Los enlaces son claramente identificados		100 %			
Los enlaces funcionan correctamente	33.3 %	66.7 %			
La aplicación se carga rápidamente (≤30 segundos)	33.3 %	66.7 %			
El uso de las imágenes es aceptable			100 %		
El uso del color es aceptable		33.3 %	66.7 %		
El diseño general es apropiado		33.3 %	66.7 %		
La interfaz es placentera		33.3 %	66.7 %		
Tiene todas las funcionalidades esperadas		33.3 %	33.3 %	33.3 %	

Pregunta	Excelente	Bueno	Neutro	Regular	Deficiente
¿Cómo califica globalmente la app?		70 %	30 %		

Anexo 5: Resultados Observación

Las secciones que siguen a continuación detallan los resultados obtenidos en cada una de las observaciones que se hizo, siguiendo la pauta que se detalló anteriormente.

Resultados Claudio Gutierrez

Nombre	Claudio Gutierrez		
Sexo	X Masculino		Femenino
Experiencia en Endpoints	Pocos Minutos	Horas	X Días

Observación o Tarea	**Logrado Fácilmente**	**Logrado con Dificultad**	**No Logrado**	**Comentarios**
Realizar Query		X		
Log In(Graph Owner)	X			
Log In	X			
Subir Grafo		X		
Update en Grafo (grafo permitido)		X		
Update en Grafo (grafo no permitido)		X		
Agregar Usuario	X			
Agregar Usuario a Grafo	X			
Cambiar Clave	X			
Log Out	X			

Resultados Camilo Garrido

Nombre	Camilo Garrido		
Sexo	X Masculino		Femenino
Experiencia en Endpoints	Pocos Minutos	Horas	X Días

Observación o Tarea	**Logrado Fácilmente**	**Logrado con Dificultad**	**No Logrado**	**Comentarios**
Realizar Query	X			
Log In(Graph Owner)	X			
Log In	X			
Subir Grafo		X		
Update en Grafo (grafo permitido)	X			
Update en Grafo (grafo no permitido)	X			
Agregar Usuario	X			
Agregar Usuario a Grafo	X			
Cambiar Clave	X			
Log Out	X			

Resultados Mauricio Quezada

Nombre	Mauricio Quezada		
Sexo	X Masculino		Femenino
Experiencia en Endpoints	Pocos Minutos	Horas	X Días

Observación o Tarea	**Logrado Fácilmente**	**Logrado con Dificultad**	**No Logrado**	**Comentarios**
Realizar Query	X			
Log In(Graph Owner)	X			
Log In	X			
Subir Grafo	X			
Update en Grafo (grafo permitido)	X			
Update en Grafo (grafo no permitido)	X			
Agregar Usuario		X		
Agregar Usuario a Grafo		X		
Cambiar Clave	X			
Log Out	X			

Resultados Daniel Hernandez

Nombre	Daniel Hernandez		
Sexo	X Masculino		Femenino
Experiencia en Endpoints	Pocos Minutos	Horas	X Días

Observación o Tarea	**Logrado Fácilmente**	**Logrado con Dificultad**	**No Logrado**	**Comentarios**
Realizar Query	X			
Log In(Graph Owner)	X			
Log In	X			
Subir Grafo	X			
Update en Grafo (grafo permitido)	X			
Update en Grafo (grafo no permitido)	X			
Agregar Usuario	X			
Agregar Usuario a Grafo	X			
Cambiar Clave	X			
Log Out	X			

Bibliografía

[1] Claudio Agostino, Ernesto Damiano, Sabrina De Capitani, and Pierangela Samarati. *A web service architecture for enforcing access control policies*. In Proceedings of 1st International Workshop on Views on Designing Complex Architectures, 2004.

[2] Ryan Ausanka-Crues. *Methods for Access Control: Advances and Limitations*. http://www.cs.hmc.edu/mike/public_html/courses/security/s06/projects/ryan.pdf. Accesado 1 September 2014.

[3] Luca Costabello, Serena Villata, Oscar Rodriguez-Rocha, and Fabien Gandon. *Access Control for HTTP Operations on Linked Data*. 10th Extended Semantic Web Conference (ESWC), 2013.

[4] David F. Ferraiolo, Janet A. Cugini, and D. Richard Kuhn. *Role- Based Access Control (RBAC): Features and Motivations*. In Proceedings of the Annual Computer Security Applications Conference, IEEE Press, Los Alamitos, Calif., 1995.

[5] Alban Gabillon and Léo Letouzey. *A View Based Access Control Model for SPARQL*. 4th International Conference on Network and System Security (NSS).IEEE, p. 105-112., 2010.

[6] Alban Gabillon and Léo Letouzey. *Two-step Role-Based Access Control method for Ontology Storage*. The 2012 World Congress in Computer Science, Computer Engineering, Applied Computing (WORLD COMP'12), 2012.

[7] Sabrina Kirrane, Ahmed Abdelrahman, Alessandra Mileo, and Stefan Decker. *Secure Manipulation of Linked Data*. ISWC 2013. Springer, Berlin Heidelberg, p.248–263., 2013.

[8] Sabrina Kirrane, Alessandra Mileo, and Stefan Decker. *Applying DAC principles to the RDF graph data model*. IFIP AICT. Springer, Heidelberg, vol. 405, p.69–82., 2013.

[9] Joon S. Park, Ravi Sandhu, and Gails-Joon Ahn. *Two-step Role-Based Access Control method for Ontology Storage*. ACM Transactions of Information and System Security (TISSEC), v.4 n.1, p.37-71, 2001.