

Lenguajes y Tecnologías de la Web Semántica

Aplicaciones a Datos Abiertos

Entrega trabajo de Laboratorio
11/12/2018

Universidad de la República
Montevideo, Uruguay, Diciembre de 2018

Ing. Gabriela Delfino
Ing. Sofía Maiolo
Ing. Esteban Soba

Índice

INTRODUCCIÓN	3
Objetivo del proyecto	3
Pasos realizados	3
Tecnologías y herramientas utilizadas	3
Componentes de la solución	4
Instrucciones de instalación	5
IMPLEMENTACIÓN DE LA SOLUCIÓN	7
1. Definición de la Ontología	7
Realidad a modelar	7
Herramienta y sintaxis	7
Clases y propiedades	7
Propiedades clave	8
Características de las Object properties	8
Relación con vocabularios ya existentes	9
Nomenclatura de URIs	9
2. Estrategia para extracción y carga automática de datos	10
2.a. Extracción de datos desde un sitio web	10
2.b. Formato de serialización de los datos	11
2.c. Creación de base de datos Stardog con distintos grafos	11
2.d. Transformar datos del grafo intermedio al grafo final	12
3. Aplicación web para carga y consulta de datos	17
3.a. Módulo para carga de datos	17
3.b. Módulo para consultas de datos	19
TRABAJOS FUTUROS	23
CONCLUSIONES	24

INTRODUCCIÓN

Objetivo del proyecto

Obtener una base de datos rdf cargada con las resoluciones tomadas en las reuniones de Consejo de Facultad de Ingeniería. Dejar disponible un servicio que permita consultar los datos de las mismas aprovechando el potencial que brinda el razonador OWL.

Pasos realizados

1. Definición de la ontología

En base a un estudio de casos de distintas resoluciones y repartidos, identificamos las entidades y relaciones de interés para este trabajo, reflejándolo en una ontología OWL.

2. Extracción automática de datos

Se desarrolló una aplicación crawler que recorre las publicaciones de repartidos en la web, parsea las páginas html y genera un archivo json-ld con los datos extraídos.

3. Creación y carga de base de datos

Creamos una base de datos en Stardog, cargando la ontología definida en un grafo y los datos extraídos en el json en otro grafo (intermedio) de la misma base. Luego transformamos, mediante sentencias sparql, los datos del grafo intermedio que se pudieron procesar automáticamente y cargamos los nuevos datos en el grafo final.

4. Desarrollo de aplicación para carga de datos

Con el fin de insertar los datos faltantes en nuestra base de datos Stardog, se desarrolló una aplicación web que brinda una interfase para la carga de dichos datos.

5. Desarrollo de módulo para consulta de datos

Dentro de la aplicación desarrollada se creó un módulo que permite realizar consultas sparql y visualizar los resultados de las mismas.

Tecnologías y herramientas utilizadas

- Se utilizó **Protégé** para la edición de la ontología.
- Se implementó una aplicación en **Scrapy (python)** para generar el json a partir de los html del sitio.
- Se utilizó **Stardog** para la creación de la base de datos, y carga de grafos a partir de la ontología y el json generados. Luego se dejó disponible como SPARQL endpoint para consultas y actualización de la base de datos.

- Como ambiente para el desarrollo y testing de las sentencias SPARQL se eligió **Stardog Studio**
- Se extendió la funcionalidad de Stardog, desarrollando una función en **Java**.
- El desarrollo de la aplicación web para carga de datos a mano se realizó en **Angular**, utilizando el protocolo **HTTP** para comunicación con Stardog y la librería **TypeAhead** para facilitar el ingreso de ciertos datos.
- Se utilizó **YASGUI** para brindar una interface para realizar consultas SPARQL (YASQE) y mostrar resultados (YASR).

Componentes de la solución

La solución desarrollada para este proyecto consta de los siguientes componentes, que se adjuntan a esta entrega y quedan disponibles en el repositorio de GitHub <https://github.com/XtebanUy/websemantica>

- **resoluciones.owl**: Ontología OWL que modela el problema
- **repartidos.json**: archivo en formato JSON-LD cargado con datos existentes en www.expe.edu.uy
- **Aplicación spider**: código fuente del crawler desarrollado en Scrapy para extracción de datos. <https://github.com/XtebanUy/websemantica/tree/master/spider>
- **stardogFunctions-0.0.1-SNAPSHOT.jar**: paquete java para extender la funcionalidad de Stardog con una nueva función de procesamiento de nombres de las personas. El código fuente para generar el jar está disponible en <https://github.com/XtebanUy/websemantica/tree/master/stardogFunctions>
- **consultasETL.sparql**: script con consultas de extracción y carga de datos desde el grafo intermedio creado a partir del JSON hacia el grafo Final.
- **Aplicación resolucioneshttp**: aplicación web desarrollada en Angular que permite la carga y consulta de datos. Utiliza **YASGUI** y librería **TypeAhead**. <https://github.com/XtebanUy/websemantica/tree/master/resolucioneshttp>
- **Carpeta consultas**, que contiene 6 archivos sparql con consultas de ejemplo que resultan interesantes para ejecutar con los datos cargados automáticamente.
- **EntregaLaboratorio.pdf**: este documento. Contiene toda la documentación del proyecto

Instrucciones de instalación

Requerimientos

Para poder ejecutar la solución implementada se debe tener previamente instalado:

- Stardog (en nuestro ambiente de desarrollo utilizamos la versión 6)
Ver https://www.stardog.com/docs/#_quick_start_guide
- Angular - <https://angular.io/guide/quickstart>
- JAVA 8 y Apache MAVEN (si se quiere compilar función que extiende Stardog)
- Python 3 y Scrapy (para compilar el crawler y generar el json)
Python 3 se puede descargar del sitio oficial <https://www.python.org/downloads/> o puede ser instalado en linux usando el gestor de paquetes del sistema operativo.

Pasos de instalación

A continuación se listan los pasos a realizar para dejar funcionando la solución:

1. Copiar los archivos:
 - resoluciones.owl y repartidos.json a la carpeta STARDOG/bin
 - stardogFunctions-0.0.1-SNAPSHOT.jar a la carpeta STARDOG/bin/server/ext

En caso de querer generar el jar, bajar el código fuente de la carpeta stardogFunctions del repositorio GitHub. Para compilar y empaquetar, ejecutar:

```
mvn compile
mvn package
```

El archivo repartidos.json se adjunta ya generado a esta entrega. Al final de este capítulo se brindan instrucciones para compilar el crawler y así poder generar un nuevo archivo.

2. Levantar el servicio Stardog
3. En línea de Comandos ejecutar la creación de la base de datos y la carga de los grafos:

```
./stardog-admin db create -n dbresoluciones
./stardog data add dbresoluciones repartidos.json
./stardog data add --named-graph http://www.fing.edu.uy/test/base/final
dbresoluciones resoluciones.owl
```

4. En Stardog o Stardog Studio, entrar a la base de datos dbresoluciones y configurar los siguientes prefijos:

```
prefix base: <http://www.fing.edu.uy/test/base/>
prefix vocab: <http://www.fing.edu.uy/test/vocab#>
prefix res: <http://www.semanticweb.org/fing/ontologies/2018/resoluciones/>
prefix resvocab: <http://www.semanticweb.org/fing/ontologies/2018/resoluciones#>
```

5. Conectado a la base dbresoluciones y con el razonador activado, abrir el script consultasETL.sparql y ejecutarlo.

6. Dejar Stardog levantado para poder consultar desde la aplicación web
7. Instalar la aplicación Angular corriendo los siguientes pasos:
 1. Copiar la carpeta resolucioneshttp en el disco duro. Se puede hacer git clone desde <https://github.com/sophiamaiolo/resolucioneshttp.git>
 2. Abrir línea de comandos y posicionarse en la carpeta
 3. Ejecutar: `npm install`
 4. Copiar la carpeta images de `node_modules\leaflet\dist` a `node_modules\yasgui-yasr\dist`
 5. Ejecutar: `ng serve --open`
9. Acceder a la aplicación en <http://localhost:4200/>

Compilación del crawler

Estas instrucciones se brindan en caso de que se quiera compilar la aplicación crawler para generar un nuevo archivo .json con datos actualizados del sitio fuente. Los pasos indicados fueron probados en sistema operativo Linux.

Es recomendable crear un environment para trabajar con una copia de Python con las librerías necesarias para el proyecto que estemos desarrollando. Ejecutar:

```
python3 -m venv /path de instalación
```

Una vez creado el environment lo activamos con

```
source /path de instalación/bin/activate
```

Luego instalamos las dependencias del proyecto:

```
pip install scrapy
```

Una vez finalizada la instalación podremos correr la spider para extraer los datos utilizando el comando:

```
scrapy crawl extraer_repartidos --nolog
```

IMPLEMENTACIÓN DE LA SOLUCIÓN

1. Definición de la Ontología

Realidad a modelar

Habiendo estudiado distintos casos publicados en www.expe.edu.uy, concluimos que la realidad a modelar tiene las siguientes características:

- Cada Repartido corresponde a una sesión del Consejo de Facultad de Ingeniería donde se tomaron distintas resoluciones. De la sesión nos interesa saber qué tipo de sesión fue (Ordinaria, Extraordinaria) y los asistentes a la misma por cada orden (Decano, Egresados, Docentes, Estudiantil).
- De cada resolución tomada en una sesión nos interesa el/los expedientes a los que está asociada y quien (persona u organización) la solicita, además de la cantidad de votos con que fue aprobada
- Además interesa saber los antecedentes, que se describen como Documentos (con un tipo y un autor asociados) y las acciones definidas y quien (persona u organización) se ve afectado o involucrado por la acción.
- Las Personas y Organizaciones son entidades que pueden participar de distintas formas de un repartido o resolución (asistentes a una Sesión, solicitantes de una resolución, autores de un documento de antecedente o involucrados en una acción)
- De las Organizaciones además nos interesa saber la dependencia institucional entre ellas.

Herramienta y sintaxis

Para la definición de la ontología utilizamos Protégé, almacenando el resultado con sintaxis RDF/XML, ya que es uno de los estándares recomendados por W3C que debe ser soportado por todas las herramientas que trabajan con OWL2, y en particular es soportado por Stardog, la herramienta elegida para este proyecto. Así obtuvimos el archivo **resoluciones.owl**.

Clases y propiedades

La ontología definida está compuesta por las siguientes clases y propiedades:

Clases (y dataProperties):

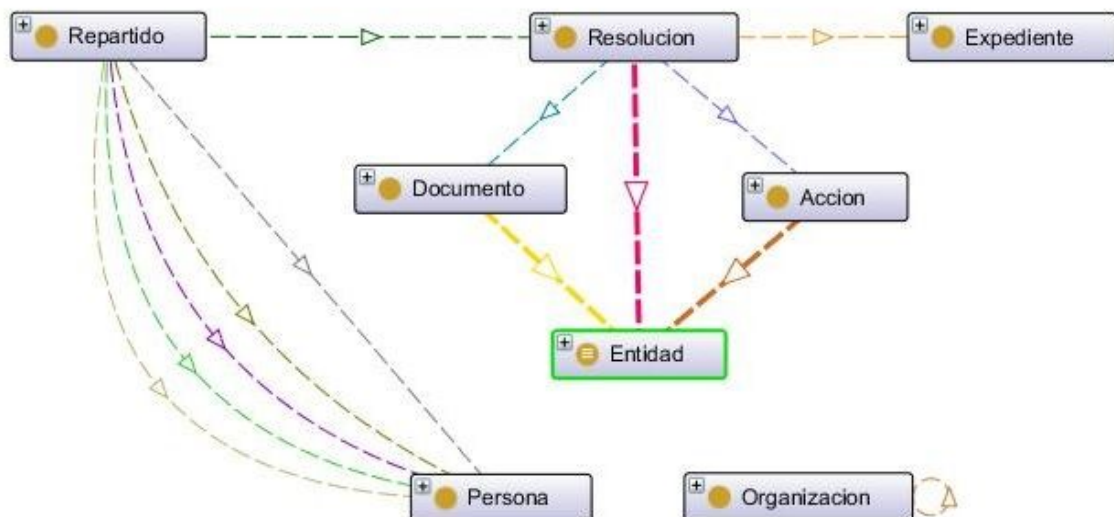
- Repartido (nroRepartido, infoSesion, fecha, cantAsistentesSesion, urlPagRepartido)
- Resolucion (nroResolucion, fecha, cantVotosResolucion)
- Documento (tipoDocumento, temaDocumento)
- Expediente (nroExpediente, uriExpediente)
- Acción (descripcionAccion)
- Persona (nombre)
- Organización (nombre, tipoOrganizacion)
- Entidad

Object Properties:

- repartidoResolucion

- asistente
 - asistenteEgresados
 - asistenteDecano
 - asistenteEstudiantil
 - asistenteDocentes
- resolucionExpediente
- resolucionSolicitante
- resolucionDocumentos
- resolucionAcciones
- accionInvolucrados
- repartidoInvolucrados
- esParteDe

Para poder visualizar mejor las relaciones definidas creamos una nueva clase llamada Entidad, que es la unión de las clases Persona y Organizacion. Se pueden visualizar estas Clases y relaciones en el siguiente esquema.



Propiedades clave

Para las clases Expediente y Repartido pudimos identificar Data properties que identifican a cada individuo: nroExpediente y nroRepartido respectivamente. Las demás clases no tienen una propiedad que las identifique, si bien el nombre de las personas y las organizaciones se ha usado en la práctica, para la carga de datos como un identificador.

Características de las Object properties

Se identificaron las siguientes características de las propiedades

- Funcional inversa:
 - repartidoResolucion
 - resolucionAcciones
 - resolucionDocumentos
- Transitiva:
 - esParteDe

A su vez se definió la propiedad `repartidoInvolucrados` como una cadena de propiedades, de forma de poder agrupar todos los involucrados en todas las acciones de cada repartido: `repartidoResolucion` o `resolucionAcciones` o `accionInvolucrados` `subPropertyOf`: `repartidoInvolucrados`

Relación con vocabularios ya existentes

Las Organizaciones y sus relaciones pueden definirse como subclases o subpropiedades del vocabulario Academic Institution Internal Structure Ontology (AIISO)

<http://purl.org/vocab/aiiso/schema#>

- `Organizacion` sería subclase de <http://vocab.org/aiiso/schema#organization>
- `esParteDe` sería subpropiedad de http://purl.org/vocab/aiiso/schema#part_of

`Personas` puede considerarse subclase de `foaf:Person`

http://xmlns.com/foaf/spec/#term_Person

Durante el desarrollo del proyecto se intentaron definir estas relaciones en Protegé, pero no se reconocían las entidades a las que se hacía referencia (mencionadas arriba). Se consiguió que reconociera los términos al importar todo el vocabulario dentro de nuestra ontología pero no nos pareció una solución adecuada, por lo que la definición de estas relaciones quedó pendiente.

Nomenclatura de URIs

Tipos de URIs

Hashtag URIs (# luego del pref): definiciones de la ontología (clases y propiedades).

303 URIs (/ luego del pref): Individuos

Prefijos:

resvocab: <http://www.semanticweb.org/fing/ontologies/2018/resoluciones#>

res: <http://www.semanticweb.org/fing/ontologies/2018/resoluciones/>

Nomenclatura definida para nombrar cada individuo:

Repartido: **Rep**NroRepartidoFaaaamdd

Resolucion: **Res**NroResolucionRNroRepartidoFaaaamdd

Documento: **Doc**STRUOID()

Accion: **Acc**STRUOID()

Expediente: **Expediente**NroExpediente

Persona: **Persona**Nombre

Organizacion: **Org**Nombre

Se buscó una nomenclatura basada en las propiedades únicas de cada individuo, de forma de poder reconstruir las URIs basados en estas propiedades, y para facilitar la lectura de los datos, sobre todo en la etapa de desarrollo del proyecto.

En el caso de Documentos y Acciones, al ser elementos que no tienen propiedades que ayuden a identificarlos,, se decidió usar STRUOID() para construir las URIs.

2. Estrategia para extracción y carga automática de datos

2.a. Extracción de datos desde un sitio web

Los repartidos se encuentran publicados en el sitio <http://www.expe.edu.uy/expe/>

En particular los repartidos de la Facultad de Ingeniería se obtienen de <http://www.expe.edu.uy/expe/resoluci.nsf/repartidosfin06?OpenView&Start=1&Count=150&Expand=1#1>

En el parámetro Count se puede especificar la cantidad de repartidos que queremos obtener. Para este ejemplo obtuvimos un listado de 150 repartidos.

La extracción de datos a partir de la información presentada en páginas web se realizó utilizando la técnica de crawling. Para esto se utilizó Scrapy <https://scrapy.org/>, que es un framework open source desarrollado en Python para realizar extracción de datos de sitios web.

A partir de los links que se obtienen de la página inicial se navega hasta los repartidos. Sobre cada repartido utilizando xpaths se extrajeron textos que representan las propiedades de los repartidos y resoluciones. Para refinar estos valores quitando información no relevante se utilizaron expresiones regulares.

Ejemplo de la página de un repartido:

<http://www.expe.edu.uy/expe/resoluci.nsf/623d08cbcb4943eb03256f3500602934/97373c6fa611ae09032582ea007ca86e?OpenDocument>

La información extraída cumple con la siguiente estructura:

Repartido

- info_de_la_sesion (infoSesion)
- fecha_hora (fecha)
- numero (nroRepartido)
- asistentes (*es un texto a procesar*)
- url (urlPagRepartido)
- texto (títulos de la página del repartido, en principio no se usan)

Resolucion (n resoluciones por repartido)

- numero (nroResolucion)
- expediente (nroExpediente, puede haber más de 1)
- texto (*texto a procesar*)

Durante el testing de esta aplicación pudimos observar que, si bien la mayoría de los datos de las resoluciones y repartidos pudieron extraerse correctamente, en algunos casos

existen errores en la información cargada (por ejemplo el texto de una resolución no aparece completo en algún caso).

Algunos de los errores detectados pudieron corregirse en sucesivas etapas de testint y corrección, pero en su mayoría se debe a que los documentos htmls de origen no poseen una estructura uniforme, sumado a que el html está mal formado, lo que no nos permitió procesar automáticamente todos los casos de forma correcta.

2.b. Formato de serialización de los datos

En una primera instancia los datos recabados se almacenaron en un archivo json sin una estructura particular. El contenido de este json debía ser cargado a una base de datos SPARQL por lo que se decidió investigar sobre JSON-LD.

JSON-LD es un formato ligero de Linked Data, que consiste en un JSON al cual se le agregan keywords que permiten darle un significado a las propiedades. La transformación para convertir la información a formato JSON-LD se realizó en el proceso de serialización de la aplicación de crawling.

La propiedad que permite realizar esto es `@context`. En esta propiedad se indica un contexto sobre el cual las propiedades tienen un valor. Dentro del contexto se agrega la información del significado de cada propiedad. Si se quiere que las propiedades tengan un prefijo común esto se realiza utilizando la propiedad `@vocab`. Con el objetivo de que las URIs relativas tengan un prefijo común se utiliza la propiedad `@base`.

Se puede utilizar la propiedad `@id` para definir el nombre del grafo donde se cargaron los datos. Y los datos en sí se incluyen en un elemento `@graph`, quedando nuestro JSON con la siguiente estructura:

```
"@context": {
  "@vocab": "http://www.fing.edu.uy/test/vocab#",
  "@base": "http://www.fing.edu.uy/test/base/",
  "@version": "1.1",
  "resoluciones": {
    "@container": "@set", "@id": "resoluciones"
  }
},
"@id": "intermedio",
"@graph": [{ ...
  ... acá van todas las propiedades extraídas de la web
}]
```

El archivo fue nombrado **repartidos.json**

2.c. Creación de base de datos Stardog con distintos grafos

Decidimos trabajar con una base de datos que llamamos **dbresoluciones**, y en esta base se tendrán dos grafos distintos

- **intermedio:** creado a partir de los datos obtenidos en el json-ld
- **final:** creado en base a la ontología definida, que luego será poblado a partir de los datos del grafo intermedio:
 - procesando automáticamente lo que se pueda
 - y utilizando una interfase web para procesar e insertar la información que no pueda cargarse automáticamente

El primer paso es crear la base de datos.

Luego se carga el archivo json-ld generado en el paso anterior. En el propio archivo json-ld se indica que se cargará en un grafo de nombre **intermedio**, agregando luego del @context la propiedad "@id": "intermedio".

El siguiente paso es crear un grafo de nombre **final** a partir de la ontología definida y almacenada en resoluciones.owl. Ejecutamos el comando:

Comando utilizados para realizar los pasos anteriores

```
./stardog-admin db create -n dbresoluciones
./stardog data add dbresoluciones repartidos.json
./stardog data add --named-graph http://www.fing.edu.uy/test/base/final dbresoluciones
resoluciones.owl
```

Otra opción para cargar los grafos, en vez de usar la línea de comandos, es mediante la herramienta Stardog Studio, la que proporciona una interface gráfica que permite seleccionar la base de datos sobre la que se quiere trabajar y el archivo a cargar.

A través Stardog Studio agregamos los siguientes prefijos a la base:

```
prefix base: <http://www.fing.edu.uy/test/base/>
prefix vocab: <http://www.fing.edu.uy/test/vocab#>
prefix res: <http://www.semanticweb.org/fing/ontologies/2018/resoluciones/>
prefix resvocab: <http://www.semanticweb.org/fing/ontologies/2018/resoluciones#>
```

2.d. Transformar datos del grafo intermedio al grafo final

Se crean las sentencias para consultar la información cargada en el grafo intermedio, transformarla e insertarla en el grafo final.

Sobre las URIs

En el grafo intermedio los individuos son identificados con la misma URI que usamos en el grafo final, pero con distinto prefijo. Al pasar de un grafo al otro, reemplazamos el prefijo base: por el prefijo res: usando la siguiente conversión

```
BIND (IRI(replace(str(?uri), str(base:), str(res:))) AS ?urif)
```

Transformaciones realizadas

1. Insertamos los repartidos y sus dataproperties
Esta sentencia es bastante directa, simplemente hubo que hacer una transformación de los prefijos de las uris.

```

INSERT { GRAPH base:final
  {?repf a resvocab:Repartido.
    ?repf resvocab:nroRepartido ?nro.
    ?repf resvocab:infoSesion ?i.
    ?repf resvocab:fecha ?f.
    ?repf resvocab:urlPagRepartido ?u
  }
}
where {graph base:intermedio
  {?rep rdf:type vocab:Repartido.
    ?rep vocab:numero ?nro.
    ?rep vocab:info_de_la_sesion ?i.
    ?rep vocab:fecha_hora ?f.
    ?rep vocab:url ?u.
    BIND (IRI(replace(str(?rep), str(base:), str(res:))) AS ?repf)
  }
}

```

2. Insertamos los asistentes por Orden por el que asisten

En esta sentencia, por cada asistente, se inserta primero la persona, su nombre y luego la relación con el repartido.

Para el armado de la sentencia, se tuvo en cuenta que hay 4 propiedades distintas que representan las relaciones, por lo que se obtuvieron todos los asistentes utilizando UNION de 4 patrones de grafo, e identificando en cada caso por qué orden asistieron, de forma de poder utilizar la propiedad correcta al insertar en el grafo final.

Función ExtraerNombre()

En el grafo intermedio las personas figuran con distintos títulos (como Ing., Br., Prof., etc) antes del nombre. Para quitar los títulos y obtener únicamente los nombres de las personas implementamos una función de Stardog. Al correr dentro de Stardog, esta función tiene como requerimiento estar impleementada en JAVA. Información adicional sobre como extender Stardog con funciones custom se encuentra en https://www.stardog.com/docs/#_query_functions.

La función se empaqueta en un jar y se copia en la carpeta STARDOG/bin/server/ext para que quede disponible para ser utilizada en Stardog. Para utilizarla en una consulta se invoca mediante `stardog:ExtraerNombre()`.

Si bien esta tarea se podía resolver utilizando sparql puro, nos resultó interesante el ejercicio de extender la funcionalidad de Stardog y al mismo tiempo aprovechar la facilidad de procesamiento de estos strings utilizando expresiones regulares en java, que nos resultó más sencillo que el uso de expresiones regulares en sparql.

```

INSERT
{
  GRAPH base:final
  {
    ?piri a resvocab:Persona.
    ?piri resvocab:nombre ?np.
    ?siri ?rel ?piri.
  }
}
WHERE {graph base:intermedio
{
  {{?s vocab:asistentes ?a.
    ?a vocab:decano ?p.
    FILTER(?p!="") }
    BIND(resvocab:asistenteDecano as ?rel)
  }
  UNION
  {{?s vocab:asistentes ?a.
    ?a vocab:ordenEstudiantil ?p.
    FILTER(?p!="") }
    BIND(resvocab:asistenteEstudiantil as ?rel)
  }
  UNION
  {{?s vocab:asistentes ?a.
    ?a vocab:ordenDocente ?p.
    FILTER(?p!="") }
    BIND(resvocab:asistenteDocentes as ?rel)
  }
  UNION
  {{?s vocab:asistentes ?a.
    ?a vocab:ordenEgresado ?p.
    FILTER(?p!="") }
    BIND(resvocab:asistenteEgresados as ?rel)
  }
  BIND(stardog:ExtraerNombre(?p) as ?np)
  BIND(IRI(concat(str(res:), "Persona", ENCODE_FOR_URI(?np))) as ?piri)
  BIND(IRI(replace(str(?s), str(base:), str(res:))) AS ?siri)
} };
```

3. Insertamos/Actualizamos propiedad cantAsistentesSesion

Al ejecutar esta sentencia debemos tener prendido el razonador ya que se utiliza la propiedad resvocab:asistente, que es inferida por sus subpropiedades.

En este caso utilizamos un DELETE INSERT, para borrar un posible registro existente que estuviera desactualizado. Se utiliza OPTIONAL, ya que queremos que la consulta devuelva la cantidad de asistentes de todos los repartidos, tengan o no ya cargada esa propiedad.

```
WITH base:final
DELETE { ?rep resvocab:cantAsistentesSesion ?c }
INSERT { ?rep resvocab:cantAsistentesSesion ?cant }
WHERE {
  {
    SELECT ?rep (count(?a) as ?cant)
    where {?rep a resvocab:Repartido.
           ?rep resvocab:asistente ?a.
          }
    group by ?rep
  }
  OPTIONAL {?rep resvocab:cantAsistentesSesion ?c}
}
```

4. Insertamos las resoluciones de cada repartido con su número

```
INSERT { GRAPH base:final
  {?resf a resvocab:Resolucion.
   ?resf resvocab:nroResolucion ?n.
   ?rf resvocab:repartidoResolucion ?resf.
  }
}
where {graph base:intermedio
  {?r vocab:resoluciones ?res.
   ?res a vocab:Resolucion.
   ?res vocab:numero ?n.
   BIND (IRI(replace(str(?res), str(base:), str(res:))) AS ?resf)
   BIND (IRI(replace(str(?r), str(base:), str(res:))) AS ?rf)
  }
}
```

5. Insertamos los expedientes de cada resolución

En este caso detectamos algunos números de expediente en el grafo intermedio que claramente no se corresponden con un número real de expediente, por lo que aplicamos un filtro para no cargar estos casos.

```
INSERT {graph base:final
  {?e2 a resvocab:Expediente.
   ?e2 resvocab:nroExpediente ?e.
   ?r2 resvocab:resolucionExpediente ?e2.
  }
}
```

```

    }
WHERE {graph base:intermedio
    {?res vocab:expediente ?e.
    FILTER (?e!="-" && CONTAINS(?e,"-"))
    }
    BIND(IRI(replace(str(?res),str(base:),str(res:))) as ?r2)
    BIND(IRI(concat(str(res:),'Expediente',encode_for_uri(?e))) as ?e2)
};

```

6. Limpieza de datos

Luego de haber realizado la carga se encontraron algunos casos, sobre todo para las personas, en que se puede mejorar la información aplicando transformaciones o insertando sentencias adicionales.

Así, por ejemplo, creamos sentencias para detectar las Personas que tienen nombre distinto pero creemos que son la misma persona, por la similitud de sus nombres (son iguales sin considerar mayúsculas/mínusculas ni tildes, o un nombre contenido en el otro). Se actualizó el nombre de estos individuos para usar un único nombre para todos los que eran similares. Y luego marcamos como equivalentes (owl:sameAs) a las personas que quedaron con el mismo nombre.

Todas las sentencias de transformación mencionadas se almacenaron en el archivo **consultasETL.sparql**, que puede ejecutarse como un script en Stardog.

Se consideró la posibilidad guardar estas sentencias a su vez como un nuevo grafo (de nombre ETL, por ejemplo) que podría ser consultado para obtener las sentencias y ejecutarlas una a una, pero debido a limitantes de tiempo y que el archivo de script resultó una solución conveniente, no se investigó más a fondo esta posibilidad.

Mediante estas transformaciones logramos insertar una buena cantidad de información, pero existe información de resoluciones que no es posible procesar automáticamente, al menos con patrones simples de procesamiento. Puede ser un trabajo a futuro investigar la aplicación de procesamiento de lenguaje natural para obtener una clasificación de los antecedentes y acciones relacionados a una resolución, así como extraer las personas u organizaciones involucradas.

3. Aplicación web para carga y consulta de datos

En este paso se desarrolla una aplicación que consulta los textos del grafo intermedio que no pudieron procesarse automáticamente, y mediante la interacción con el usuario, completa los datos de las resoluciones que no pudieron cargarse en el paso anterior.

Esta aplicación permite también la realización de consultas sobre el grafo Final.

3.a. Módulo para carga de datos

Para el desarrollo de esta parte se creó una aplicación web en Angular, donde se permiten las siguientes funcionalidades:

1. Listado de todos los repartidos ingresados en el grafo Final

Expe Semantico Home Query		
Número	Uri Repartido	
17/15	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep17/15F20150813	
18/15	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep18/15F20150903	
14/15	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep14/15F20150716	
16/15	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep16/15F20150813	
19/15	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep19/15F20150910	
15/15	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep15/15F20150730	
21/15	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep21/15F20151008	
24/15	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep24/15F20151126	
12/15	http://www.semanticweb.org/fing/ontologies/2018/	

2. Dado un repartido seleccionado, se muestran los datos de dicho repartido y se listan las resoluciones asociadas. De las resoluciones se muestra la cantidad de

antecedentes y acciones relacionados, de forma de poder visualizar rápidamente si una resolución ya fue procesada.

Expe Semantico Home Query		
Número	Uri Repartido	Repartido: 17/15 -- 2015-08-13T21:00:00 http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep17/15F20150813
17/15	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep17/15F20150813	Ocultar o Mostrar Resoluciones
18/15	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep18/15F20150903	
14/15	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep14/15F20150716	
16/15	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep16/15F20150813	

3. Dada una resolución seleccionada, se muestra el texto completo, obtenido desde el grafo Intermedio, y se permite que el usuario:
 - a. Ingrese en el grafo final una nueva Persona
 - b. Ingrese en el grafo final una nueva Organización e indicar que es dependiente de otra ya existente
 - c. Ingrese en el grafo final un nuevo Documento
 - d. Ingrese en el grafo final una nueva Acción

Número	Uri Repartido	Repartido: 14/15 -- 2015-07-16T19:00:00 http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep14/15F20150716
17/15	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep17/15F20150813	Ocultar o Mostrar Resoluciones
18/15	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep18/15F20150903	
14/15	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep14/15F20150716	
16/15	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep16/15F20150813	
19/15	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep19/15F20150910	
15/15	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep15/15F20150730	
21/15	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep21/15F20151008	
24/15	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep24/15F20151126	
12/15	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep12/15F20150618	
20/15	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep20/15F20151001	
10/15	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep10/15F20150521	

Repartido: 14/15 -- 2015-07-16T19:00:00
http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Rep14/15F20150716

Ocultar o Mostrar Resoluciones

Datos de la Resolución:
http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Res1701R14/15F20150716

Texto de la Resolución:

(Exp. S/Nº) - Aprobar lo actuado por la señora Decana que, de acuerdo a las atribuciones conferidas por el art. 42 inc. e) de la Ley Orgánica, resolvió conceder licencia por la sesión del día de la fecha a los señores consejeros del orden docente Alejandro Romanelli, Daniel Ariosa, Luis Teixeira, Álvaro Giusto y Arturo Lezama, y convocó a los suplentes preferenciales. (10 en 10)

Nueva Persona Nueva Organización

Tema Documento
Ingrese el tema del documento

Tipo Documento
Ingrese el tipo del documento

Autor Documento
Ingrese el nombre de autor del documento

Ingresar Documento

Tema Acción
Ingrese el tema del documento

Involucrado Acción
Ingrese el nombre de autor del documento

Ingresar Acción

Para el desarrollo de la aplicación se emplearon las siguientes tecnologías.

Protocolo HTTP de Stardog

Para la realización de las consultas sobre los grafos, y las correspondientes inserciones de datos, se utiliza el protocolo HTTP de Stardog. Mediante el uso de dicho protocolo se permite la realización de consultas SPARQL sobre el protocolo HTTP.

Dentro de la aplicación generada en Angular, se desarrollaron servicios que encapsulan todos los llamados HTTP que efectúan las diversas consultas.

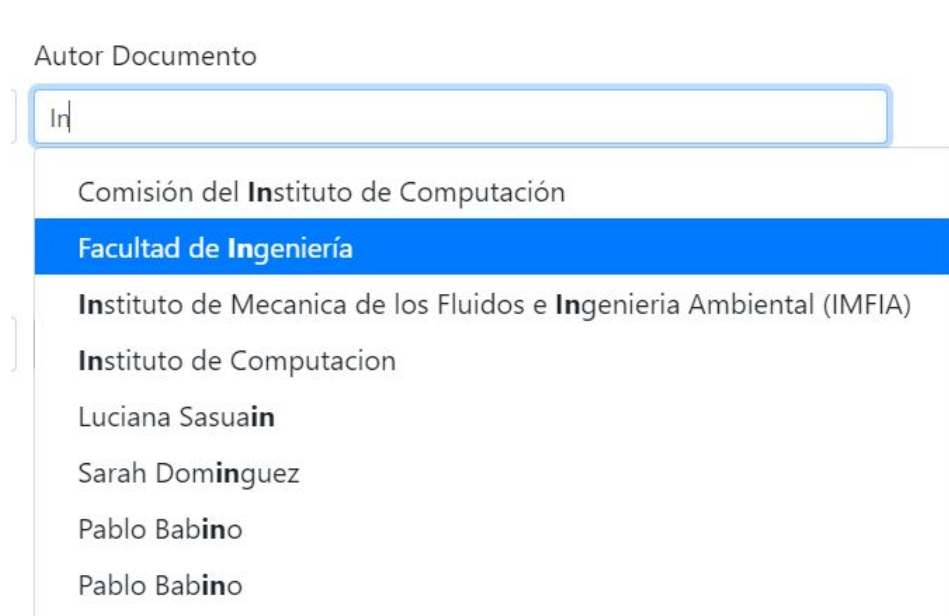
Por más detalles: https://www.stardog.com/docs/#_network_programming

Uso de Typeahead

Con el fin de facilitar la inserción de documentos, personas, acciones y organizaciones, se utiliza la librería Typeahead.

Mediante su uso, se muestran los datos ya existentes en la base de datos.

Por ejemplo, al ingresar el autor de un nuevo Documento, se despliegan las personas u organizaciones ya existentes en el grafo final:



A screenshot of a web application interface showing a Typeahead dropdown menu. The label 'Autor Documento' is positioned above a text input field. The input field contains the text 'In'. Below the input field, a dropdown menu is open, displaying a list of suggestions. The suggestions are: 'Comisión del Instituto de Computación', 'Facultad de Ingeniería' (highlighted in blue), 'Instituto de Mecanica de los Fluidos e Ingeniería Ambiental (IMFIA)', 'Instituto de Computacion', 'Luciana Sasuaín', 'Sarah Domínguez', 'Pablo Babino', and 'Pablo Babino'.

Por más detalles: <https://ng-bootstrap.github.io/#/components/typeahead/examples>

3.b. Módulo para consultas de datos

Se utiliza YASGUI para embeber un editor SPARQL en la aplicación web y mostrar los resultados de las consultas efectuadas.

YASGUI posee dos componentes: YASQE que permite embeber el editor, y YASR que permite mostrar de forma amigable los resultados obtenidos.

Esta herramienta efectúa las consultas aprovechando el protocolo HTTP de Stardog mencionado anteriormente.

De esta forma, se brinda una herramienta potente para efectuar todo tipo de consultas sobre el grafo final.

Por más detalles:

<http://yasqe.yasgui.org/>

<http://yasr.yasgui.org/>

The screenshot shows the YASGUI web interface. At the top, there's a navigation bar with 'Expe Semantico', 'Home', and 'Query'. Below it, a SPARQL query is entered in a text area:

```
1 prefix base: <http://www.fing.edu.uy/test/base/>
2 prefix vocab: <http://www.fing.edu.uy/test/vocab#>
3 prefix res: <http://www.fing.edu.uy/resoluciones/>
4 prefix resvocab: <http://www.semanticweb.org/fing/ontologies/2018/resoluciones#>
5 select * where { GRAPH base:final { ?a resvocab:resolucionDocumentos ?b.}}
```

Below the query area, there are tabs for 'Table', 'Response', 'Pivot Table', 'Google Chart', 'Geo', and a download icon. The 'Table' tab is selected, showing a table with 12 entries. The table has two columns, 'a' and 'b', and 7 rows of data. The first row shows two URIs, and the subsequent rows show various URIs and UUIDs.

a	b
http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Res9R118F20180222	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Doc1R118R9
http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Res9R118F20180222	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Doc2R118R9
http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Res9R118F20180222	http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Doc3R118R9
http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Res2148R17/15F20150813	urn:uuid:2f2613ec-b2e5-460a-a0ee-77762ac024d7
http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Res2147R17/15F20150813	4dd2df42-e9ef-4c43-bc39-bf49d3d38de
http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Res2147R17/15F20150813	6d152775-81ea-45c0-acb5-678c27e87e8b
http://www.semanticweb.org/fing/ontologies/2018/resoluciones/Res2147R17/15F20150813	13be656d-icef-4afc-9d27-ddc7f1741257

Propuesta de consultas a realizar

A continuación se listan algunas consultas que resultan interesantes para este ejemplo, sacando provecho de la potencia del razonador owl y de la información ya procesada automáticamente.

Todas las consultas se adjuntan a esta entrega como un archivo .sparql. Aquí copiamos algunas de ejemplo.

Sobre los asistentes:

1. Asistentes por órdenes distintos

Lista los nombres de las personas que asistieron a sesiones del Consejo en representación de órdenes distintos, y a cuantas sesiones asistieron por cada orden.

2. Cantidad de sesiones por asistente y datos relacionados

Lista todas las personas que asistieron a una sesión, agrupadas por el orden por el que asistieron. De cada persona se indica la cantidad de sesiones a las que asistió por ese orden y la primera y última fecha de participación.

```
select ?nom ?orden (count(?rep) as ?c) (MIN(?f) as ?ini) (MAX(?f) as ?fin)
{graph base:final
  {?rep ?orden ?p.
   ?p a resvocab:Persona.
   ?rep resvocab:fecha ?f.
   ?p resvocab:nombre ?nom.
   FILTER(?orden in (resvocab:asistenteDecano,resvocab:asistenteDocentes,
resvocab:asistenteEstudiantil, resvocab:asistenteEgresados))
  } } group by ?nom ?orden
order by ?orden DESC(?c)
```

Notar que si en esta consulta se agrega la uri de la persona (y no solo el nombre), va a devolver resultados distintos con el razonador prendido que cuando está desactivado, porque hay Personas relacionadas por owl:sameAs.

3. Cantidad de asistentes promedio por año y por tipo de sesión

```
SELECT ?a ?i (ROUND(AVG(?c)) as ?prom)
where {graph base:final
  {
    ?rep resvocab:cantAsistentesSesion ?c.
    ?rep resvocab:infoSesion ?i.
    ?rep resvocab:fecha ?f.
    BIND(STRDT(?f,xsd:dateTime) as ?ff)
    BIND(YEAR(?ff) as ?a)
  }
} group by ?a ?i
order by ?a ?i
```

Sobre los expedientes:

4. Cantidad de resoluciones por expediente

Para cada expediente se muestra la cantidad de resoluciones con las que está relacionado. Se limita el resultado para aquellos expedientes que estén asociados a más de 5 resoluciones.

```
select ?nroExp (count(?r) as ?cantRes)
{graph base:final
  {?r resvocab:resolucionExpediente ?e.
   ?e resvocab:nroExpediente ?nroExp}
} group by ?nroExp
HAVING (?cantRes>5)
order by DESC(?cantRes)
```

5. Resoluciones asociadas al expediente con más apariciones

Se obtiene el expediente con más apariciones en distintas resoluciones y se listan las resoluciones a las que está asociado, junto a la fecha de cada una.

6. Resoluciones que no están asociadas a ningún expediente

```
select distinct ?f ?rep ?n
from base:final
{
    ?r a resvocab:Resolucion.
    ?rep resvocab:repartidoResolucion ?r.
    ?rep resvocab:fecha ?f.
    ?r resvocab:nroResolucion ?n.
    FILTER NOT EXISTS {?r resvocab:resolucionExpediente ?e}
} order by ?f
```

TRABAJOS FUTUROS

Se plantean a continuación lineamientos que permitirían enriquecer la solución construida.

Ontología:

- Mejorar el diseño de la ontología. Por ejemplo: “generalizando ” los conceptos de Antecedentes y Acciones, de forma que sea más simple la edición de los mismos a nivel de la UI.
- Integrar otros vocabularios ya existentes. Se intentó utilizar otros vocabularios como foaf y aiiso, pero no se consiguió. Sería interesante volver sobre este punto y utilizarlos, para aprovechar la potencia de estos vocabularios.

Extracción de los datos:

- Utilizar elementos de NLP para extraer con mayor precisión los datos. Actualmente se utilizan expresiones regulares, pero sería interesante aprovechar la potencia de NLP para obtener mejores resultados.
- Utilizar ML para identificar personas, organizaciones, acciones, etc. De esta forma, se podría automatizar aún más la carga de los datos.
- Automatizar el proceso de extracción cada X tiempo. Se debería automatizar la extracción según la frescura de los datos que se desee tener.

ETL:

- Automatizar la ejecución de la ETL. Junto con la automatización del proceso de extracción se debería automatizar la ejecución de la ETL.
- Agregar métricas y tareas de calidad de datos. Sería interesante contar con métricas relativas a la calidad de los datos, y efectuar tareas sistemáticas con el fin de mejorarlas.
- Integrar otras fuentes de datos de interés para sacar mayor provecho a cuestiones interesantes como el uso del “same as”. Actualmente se trabaja solamente con los datos de Expe. Una línea a seguir sería integrar estos datos con otras fuentes y así aprovechar una de las cuestiones más interesantes de la web semántica como lo es la integración de datos. Algunas fuentes interesantes podrían ser: información de los docentes, datos de la prensa, etc.

Aplicación web:

- Mejorar la usabilidad de la aplicación. Se deberían realizar una mejora del diseño de la aplicación, orientada a mejorar cuestiones de usabilidad. Por ejemplo: agregar paginado y búsqueda en las tablas, agregar elementos que brinden ayuda (tooltips)
- Incorporar algunas funcionalidades faltantes: ver los antecedentes y acciones ingresados para una resolución, ingresar más de un involucrado en una acción, ingresar para una resolución los datos de cantidad de votos y entidad que solicitó su tratamiento en la sesión.
- Generar el HTML de los formularios de ingreso de datos, automáticamente a partir de la estructura de la ontología.
- Presentar consultas “listas para usar” a modo de ejemplo.
- Generar un Dashboard con indicadores de interés. Para el desarrollo de este Dashboard se podría usar librerías javascript como D3.js, Morris.js, Handsontable.js etc.
- Sacar más provecho del razonador en las consultas.

CONCLUSIONES

- Se cumplieron los objetivos planteados al inicio del proyecto. Para cumplirlos nos interiorizamos en diferentes aspectos de la web semántica: diseño de la ontología, carga de datos, ejecución de consultas SPARQL, uso del razonador, etc.
- A lo largo del proyecto, se superaron las dificultades encontradas relativas a:
 - extracción de datos
 - procesamiento de nombres, títulos, etc.
 - diseño de la ontología → pensar de una forma menos “estructurada”
 - uso extensivo de SPARQL para la realización de consultas y desarrollo de la UI
- Se trabajó con un “stack” de tecnologías bien amplio, que incluye tecnologías propias del mundo de la web semántica (Protegé, Stardog, etc.) y otras más generales (uso de Scrappy, librerías javascript, etc.)
- Subestimamos la dificultad de la extracción de los datos, lo cual nos insumió más tiempo del planificado y no pudimos focalizarnos tanto en el uso del razonador y de cuestiones interesantes de OWL.