Proyecto I

AllegroGraph

Integrantes:
Laura Arango M.
Tania C. Obando S.
Verónica Tofiño R.

Profesora:
María Constanza Pabón

Pontificia Universidad Javeriana Cali
Santiago de Cali, Colombia
2018-09-29

Informe técnico

Proceso de instalación:

Nota: La carpeta del servidor donde se instaló AllegroGraph fue /home/bd/tmp/ag6.44

1. Descargar y descomprimir los archivos de instalación comprimidos (.tar.gz) de la página de AllegroGraph.
2. Instalar los archivos corriendo la línea de comando agraph-6.4.4/install-agraph /home/bd/tmp/ag6.44 Esto abrirá el instalador de AllegroGraph para configurarlo.
   Si los pasos anteriores se ejecutaron correctamente debería aparecer lo siguiente en pantalla, donde los scripts de configuración preguntan varias cosas.

```
Welcome to the AllegroGraph configuration program.  This script will
help you establish a baseline AllegroGraph configuration.

You will be prompted for a few settings.  In most cases, you can hit return
to accept the default value.

Location of configuration file to create:
[/home/bd10/tmp/ag6.4.4/lib/agraph.cfg]:
Directory to store data and settings:
[/home/bd10/tmp/ag6.4.4/data]:
/home/bd10/tmp/ag6.4.4/data does not exist.
Would you like me to create it?:
[y]: y
Directory to store log files:
[/home/bd10/tmp/ag6.4.4/log]:
/home/bd10/tmp/ag6.4.4/log does not exist.
Would you like me to create it?:
[y]: y
Location of file to write server process id:
[/home/bd10/tmp/ag6.4.4/data/agraph.pid]:
Port:
[10035]:
User to run as:
[agraph]:

User 'agraph' doesn't exist on this system.
Create agraph user:
[y]: y

Now you must set up an initial user account for AllegroGraph.  This
account will have "super user" privileges in AllegroGraph.

SuperUser account name:
[super]: superu
SuperUser account password:
SuperUser account password (again):
Instance timeout seconds:
[604800]:




/home/bd10/tmp/ag6.4.4/lib/agraph.cfg has been created.

If desired, you may modify the configuration.  When you are satisfied,
you may start the agraph service.
```

3. Una vez que aparezca que se creó el archivo .cfg se puede iniciar el demonio corriendo el siguiente comando:
./home/bd/tmp/ag6.4.4/bin/agraph-control -–config /home/bd/tmp/ag6.4.4/lib/agraph.cfg start

Como ya se tendría el servidor allegro corriendo, sigue instalar el AllegroGraph Python Client (100.0.5.dev0), en cualquier equipo con python 2.7+ se escribe en terminal el siguiente comando:

```
pip install agraph-python
```

Luego se corre el siguiente script para comprobar si el cliente de python se puede conectar con el servidor AG (cambiando los valores de host, port, user y password).

```
from franz.openrdf.connect import ag_connect
with ag_connect('repo', host='HOST', port='PORT',
                user='USER', password='PASS') as conn:
    print conn.size()
```

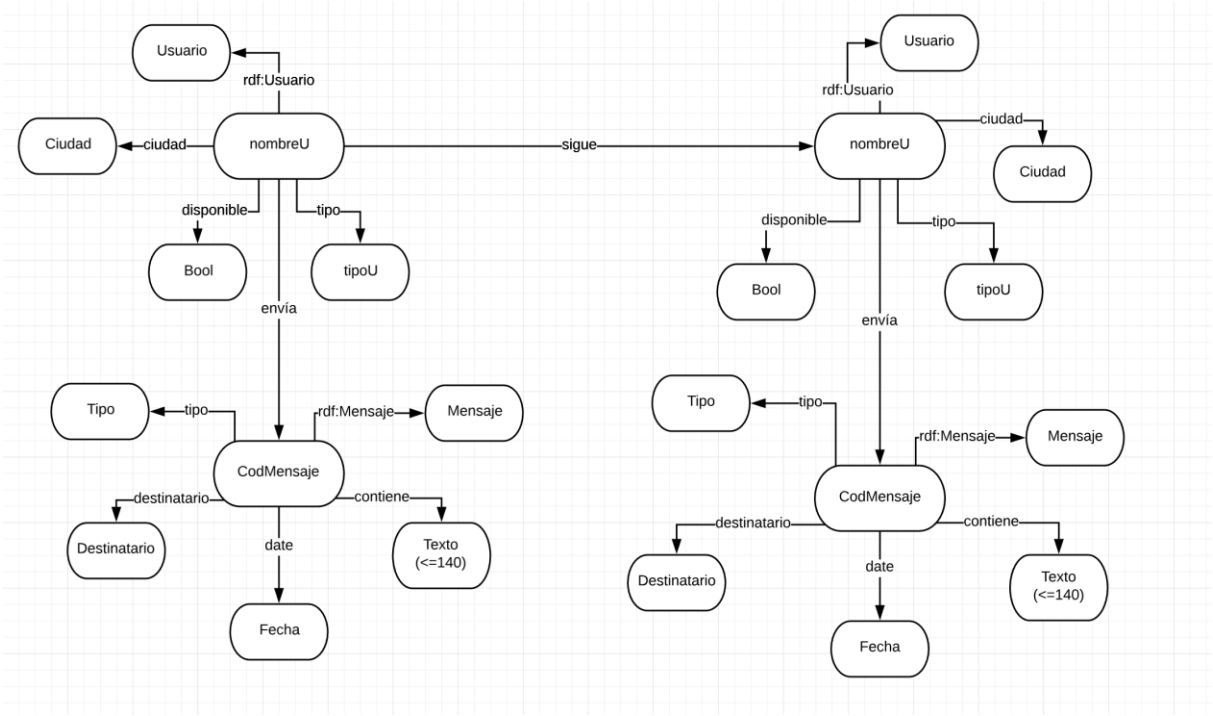
Librerías necesarias:

- *#coding=utf-8*: Utilizar caracteres especiales.
- *from franz.openrdf.sail.allegrographserver import AllegroGraphServer*: es un objeto que representa un servidor remoto allegropgraph en el internet. Es usada para acceder a los catálogos en el servidor.
- *from franz.openrdf.query.query import QueryLanguage*: Permite implmentar manipulaciones variadas sobre los datos con la consulta y la fuente.
- *from franz.openrdf.repository import Repository*: Contiene los datos RDF que pueden ser consultados y actualizados dentro de un repositorio.
- *from franz.openrdf.connect import ag_connect:* Crea una conexión a un repositorio AG.
- *from franz.openrdf.vocabulary import RDF*: Un simple pero poderoso lenguaje para representar información en forma RDF.
- *from franz.miniclient import repository*: Contiene los datos RDF que pueden ser consultados y actualizados dentro de un repositorio.
- *import os, urllib, datetime, time*

Sobre AllegroGraph:

AllegroGraph (AG) es considerada una Base de Datos RDF la cual está diseñada para almacenar y recuperar RDF triples, un formato estándar para los datos vinculados, a través de consultas semánticas. Por esta razón para guardar algo en AG es necesario escribirlo en forma de tripleta la cual se divide en **O**bjeto **P**redicado y **S**ujeto.

Modelo de datos en forma de Grafo:

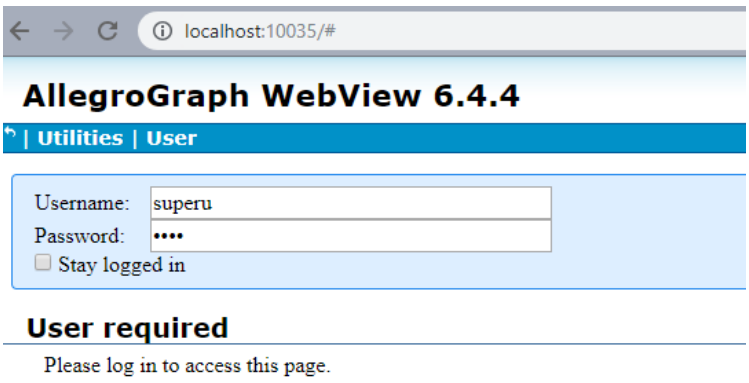


Programa en Python para crear una base de datos

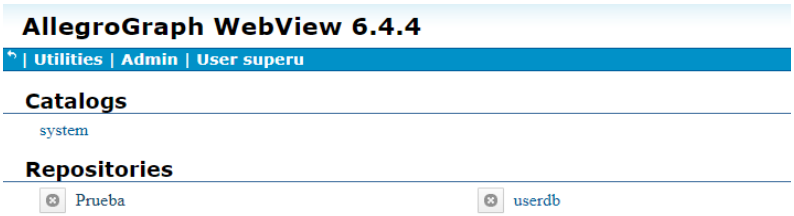
El primer programa que se corrió fue un pequeño script que creaba un repositorio llamado “userdb” en la Web y a la cual se podía acceder ingresando la contraseña y el super usuario creados en el archivo de configuración.

```
from franz.openrdf.connect import ag_connect
with ag_connect('userdb', host='localhost', port='10035',
                user='superu', password='1234') as conn:
    print conn.size()
```

Programa corrido desde consola. Nombre de la base creada Userdb.

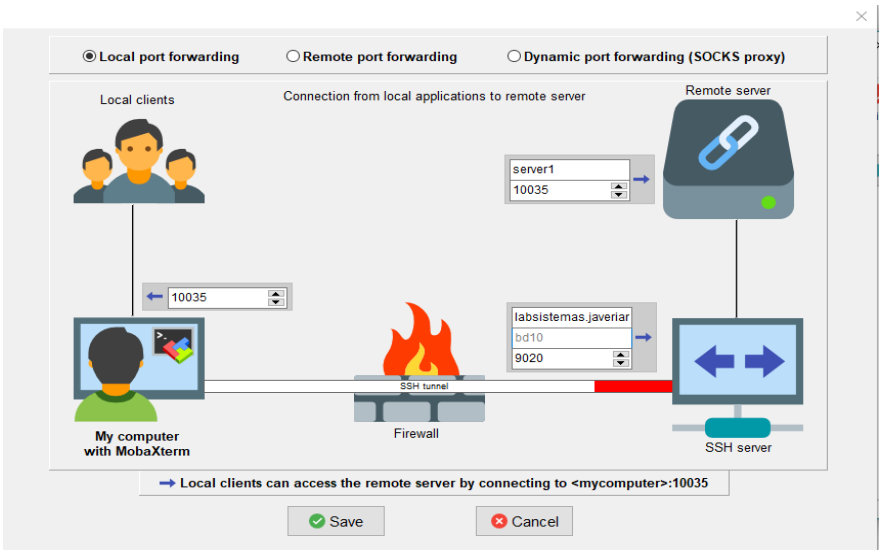


Dirección Web y donde se va a mostrar el repositorio creado.



Base de datos creada con éxito.

Nota:
Si el servidor de AllegroGraph se instaló en Linux se podrá ingresar a la página en internet con el enlace brindado para ver todo lo que se crea y se monta en el repositorio. Para poder ver la página creada desde un computador Windows se necesita hacer un túnel para poder acceder a dicha dirección.



Túnel creado con MobaXterm en Windows para poder ver lo que estaba montado en el repositorio.

Programa creado en Python para poder cargar tripletas en el repositorio (Creación de base de datos, carga de datos y consultas)

El programa consiste en el llamado de dos funciones.

- La primera función es para conectarse al repositorio creado en la Web.

```
#Cada repositorio es una coleccion de tripletas para el mismo grafo

def obtenerRepositorio(repositorio):
    serverAG = AllegroGraphServer('localhost', port=10035, user='superu', password='1234')
    catalogo = serverAG.openCatalog('')
    Repositorio = catalogo.getRepository(repositorio, Repository.ACCESS)
    Repositorio.initialize()
    return Repositorio
```

Le entra como parámetro el nombre del repositorio al que se quiere acceder. Se declaran los datos (host, puerto, usuario y contraseña) para que se pueda conectar al servidor de AllegroGraph a la cuenta creada.

Cuando se ejecute la función getRepository se le pasa el nombre del repositorio y posterior se pone o Repository.RENEW si se desea limpiar lo que hay en el repositorio y dejarlo vacío o Repository.ACCESS si se desea solo acceder al repositorio.

- La segunda función es para asignar URI[1] a cada sujeto y a cada relación. También se crean literales que posteriormente se tratarán como nodos en un modelo lógico. En la misma función se arman y suben las tripletas al repositorio al que se conectó. "Agregar" es la función que permite llenar nuestra base de datos/repositorio.

```
def agregar(repository):
    ##CONEXIÓN Y DEFINICIÓN DE NAMESPACE
    repositorio = obtenerRepositorio (repository)
    conn = repositorio.getConnection ()
    namespace_ = "http://example.org/people/"

    null = conn.createURI(namespace=namespace_,localname="Null")

    ##DEFINICIÓN DE URIS DE USUARIOS
    user = conn.createURI(namespace=namespace_,localname="User")
    lulu = conn.createURI(namespace=namespace_, localname="Lulu")
    vivi = conn.createURI(namespace=namespace_,localname="Vivi")
    nunu = conn.createURI(namespace=namespace_,localname="Nunu")


    ##DEFINICIÓN DE LAS RELACIONES
    seguir = conn.createURI(namespace=namespace_, localname="sigue-a")
```

Aunque al principio se ponía toda la URI para cada sujeto/predicado creado, como por ejemplo: user = conn.createURI("http://example.org/people/User") al agregar las tripletas salía error de sintaxis por lo que se decidió separar la extensión inicial y el nombre de la URI en namespace y localname.

```
##CREACIÓN DE LOS NODOS O SUJETOS
lunode = conn.createLiteral("Lulu")
vinode = conn.createLiteral("Vivi")
```

Para crear los sujetos es con la función de createLiteral() y como parámetro le entra el nombre que se le dará al sujeto.
Una vez creados los objetos, predicados y sujetos se añade la tripleta completa al repositorio con la función add (). Los sujetos entran como literales por ser más optimo así el realizar consultas con leídas de teclado con el nombre del usuario a buscar. Sin embargo, a cada usuario que es el sujeto de la tripleta se le creó tanto URI como literal.

[1] URI: es una cadena de caracteres diseñada para la identificación inequívoca de recursos y la extensibilidad a través del esquema de URI. Dicha identificación permite la interacción con representaciones del recurso a través de una red, típicamente la World Wide Web, usando protocolos específicos

```
grafo = conn.createURI(namespace=namespace_,localname="Usuarios")

##AGREGAR LA TRIPLETA AL REPOSITORIO
conn.add(lunode,seguir,vinode,grafo)
conn.add(vinode,seguir,lunode,grafo)


##AGREGAR RELACIONES USUARIO
conn.add(lunode,RDF.TYPE,usernode,grafo)
conn.add(vinode,RDF.TYPE,usernode,grafo)
```

La URI “grafo” se creó para que las tripletas sepan a qué grafo van las relaciones.
A todos los objetos se les asigna el Tipo o clase a la que pertenecen. En este caso es Usuario.

s	p	o
"Lulu"	rdf:type	"User"
"Vivi"	rdf:type	"User"
"Nunu"	rdf:type	"User"
"Lulu"	sigue-a	"Vivi"
"Vivi"	sigue-a	"Lulu"

Resultado en el repositorio userdb si no hay errores cuando se compila el programa en la consola.

Programa creado en Python para consultar datos ya creados en el repositorio

Lo primero siempre es llamar a la función que obtiene el repositorio para poder conectarse al repositorio creado en la Web.
A la función le entra como parámetro el nombre del repositorio donde se quiere consultar, la consulta con la que se realizará la consulta y si lo que desea obtener es el sujeto, el predicado o el objeto.

Nota: ConsultarDatos es una función que se llama dentro de otras funciones. No se llama nunca desde el main.

```
def consultarDatos(rep,query_string,tipo):
    repositorio = obtenerRepositorio(rep)
    conn = repositorio.getConnection()
    tuple_query = conn.prepareTupleQuery(QueryLanguage.SPARQL,query_string)
    result = tuple_query.evaluate()
    lista = []
    with result:
        for binding_set in result:
            s = binding_set.getValue("s")
            p = binding_set.getValue("p")
            o = binding_set.getValue("o")
            #print("%s" %(s))

            if tipo=="s":
                #print("%s" %(s))
                lista.append(s)
            if tipo=="o":
                #print("%s" %(o))
                lista.append(o)
            #print("%s %s %s " %(s,p,o))
    return lista
```

La función prepareTupleQuery() organiza la consulta ingresada en lenguaje SPARQL que es el lenguaje que trabaja AllegroGraph. Si no genera error procede a evaluar la consulta conn.evaluate() y luego por medio de un ciclo se saca el sujeto, el predicado y el objeto de la tripleta. Luego se pregunta qué es lo que se genera regresar de la consulta se guarda en la lista por si hay múltiples encuentros por parte de la consulta y se retorna la lista.

Querys o Lenguaje de consulta RDF

Es un lenguaje de computadora, específicamente un lenguaje de consulta para bases de datos capaz de recuperar y manipular datos almacenados en el formato RDF. SPARQL se ha
convertido en el lenguaje de consulta RDF estándar.

```
def getFollowers():
    nombre=str(raw_input("Porfavor ingrese nombre de usuario: "))
    consultarDatos("Prueba",'SELECT ?s ?p ?o  WHERE{?s <http://example.org/people/sigue-a> '+'"'+nombre+'"'+' .}',"s")
```

Ejemplo de una consulta en AllegroGraph:

La consulta por realizar es devolver todos los usuarios que siguen al usuario ingresado por parámetro. El formato básico de la consulta es SELECT (lo que se quiere retornar) WHERE {?s (Cualquier Sujeto) <la URI de seguir>(al querer buscar a los usuarios que siguen al usuario ingresado) y el nombre del Usuario (el objeto)}.

Webgrafía:

Franz Inc. (2018). *franz.com.* Obtenido de AllegroGraph Python API Reference: https://franz.com/agraph/support/documentation/6.4.0/python/api.html

Franz.inc. (07 de Agosto de 2018). *AllegroGraph Python client.* Obtenido de AllegroGraph Python client Documentation, Release 100.2.1.dev0: https://media.readthedocs.org/pdf/agraph-python/latest/agraph-python.pdf

Franz.inc. (12 de Septiembre de 2018). *franz.com.* Obtenido de AllegroGraph 6.4.4 Server Installation: https://franz.com/agraph/support/documentation/current/server-installation.html