Agent Server: Semantic Agent for Linked Data

Teofilo Chambilla[1,a], Claudio Gutierrez[2,b]

[1]UTEC - Universidad de Ingeniería y Tecnología, [2]Department of Computer Science, Universidad de Chile

[a]tchambilla@utec.edu.pe, [b]cgutierr@dcc.uchile.cl

Abstract. The demo features Agent Server, a web platform allowing fully distributed and decentralized querying on the Web of Linked Data. It works under the REST principles, and is lightweight and provides a safe environment in which users can develop and deploy software agents for web computing and Semantic Web, independent of their own computing devices and can run indefinitely. For this case study, We develop NautiLOD in a fully distributed version using reactive agent architectures.

Keywords: Agents, Linked Data, NautiLOD.

1 Introduction

In the literature, there are different approaches to address the navigation on the Linked Open Data (LOD). Among them is the language NautiLOD[1,2], a declarative language designed to specify navigation patterns in LOD. This language is based on regular expressions on predicates RDF intertwined with tests of the type ASK SPARQL queries issued on the RDF resources present at each node (server). There are two tools that implement the language NautiLOD related to our work. The first is the SWPORTAL [3], a Web platform where expressions can be processed by NautiLOD agents. A personal agent navigates NautiLOD semantic data sources for relevant information and provides notifications of results directly to user by email. The second tool is the SWGET [4] application fully developed in JAVA. For the approach we are presenting here, it is relevant to highlight that both SWPORTAL and SWGET applications are currently implemented in a centralized form. That is, to process a NautiLOD expression, a central node must make successive requests, using only the GET method of the HTTP protocol, to the different SPARQL Endpoints for information.

We develop a fully distributed version using *reactive agent architectures*[7] and this work is an extended of the presented case study in *"specification language for delegating tasks to the environment of the Web"*[6]. For this, four servers have been configured on the Microsoft Azure platform in which we replicated the data base of SPARQL Endpoints dbpedia.org, freebase.org, geonames.org and yago.org respectively (Fig. 1).

2 Architecture for Distributed NautiLOD

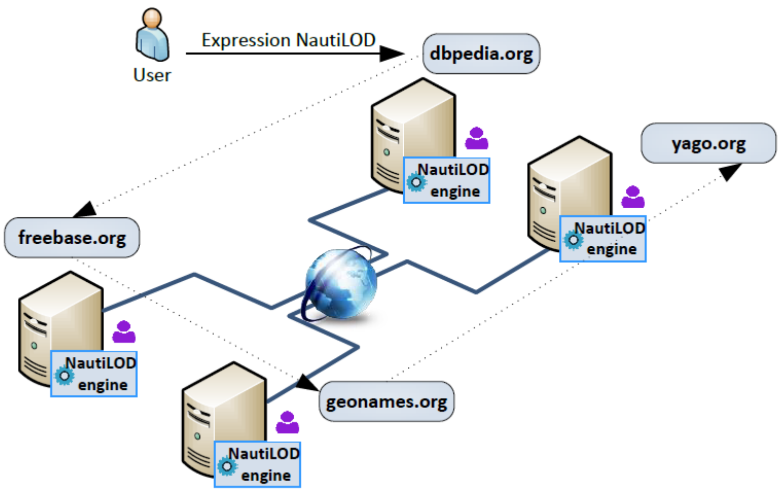
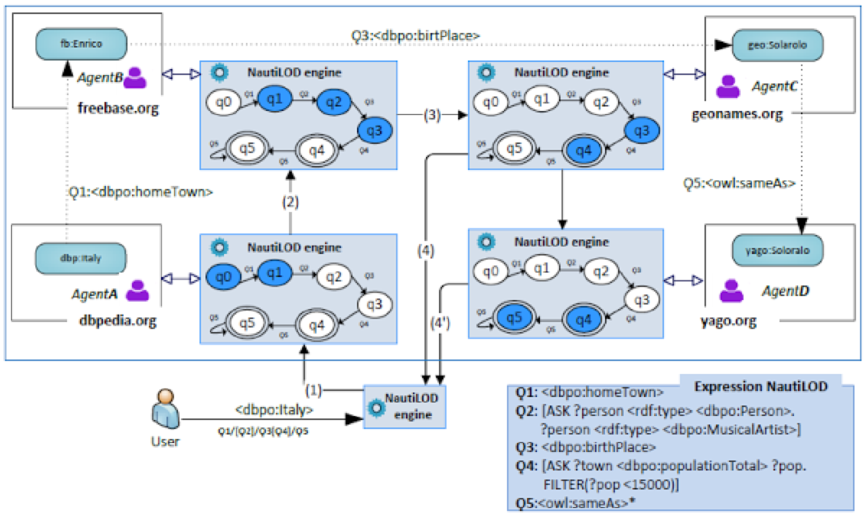


Fig. 1. Distributed NautiLOD

For allowing the agents to develop their capacities for processing task in LOD is needed a high level of communication infrastructure between platforms. The Agent Communication Language (ACL) established by FIPA would be the most appropriate for it. On the other hand, an ideal platform for MultiAgent systems would be JADE [5] because it has as characteristic a good communication infrastructure. However, although it supports the HTTP protocol communications, is not completely oriented to the Web environment. For this reason, We develop a dedicated platform called Agent Server. The platform is described in https://github.com/tchambil/agent-server. It is based entirely on the Web architecture and was developed with features of a REST API. The Agent Server platform incorporates relevant information and functions for managing agents from the behavior and life cycle point of view.
The architecture of this platform includes the Message Transport Protocol Module for reliable processing of messages, which uses the ACL specified by FIPA and managed by the methods GET, POST, PUT and DELETE of the HTTP communication protocol. Likewise Agent Manager incorporates the module responsible for the management of the platform and agents. All agents in the Agent Server platform have the same actions implemented, thus allowing to develop their capabilities in a uniform manner in cotrolled environments. Agent Manager, it is responsible for managing the life cycle, behavior and the definition of knowledge (skills) agents; Main Manager, main module of the platform, responsible for managing the operation of the Platform; Web Controller is responsible for managing the interaction of events by users, handles HTTP requests using the JSON format and the choice of it is due to both, the simplicity of their implementation; Persistence, responsible for stor-

ing all necessary information from agents, definitions, messages and the properties required for the operation of the platform. Agent Server is an extension of "Agent-Server-Stage-0".

In http://agentserver.herokuapp.com/ can be found an implementation of an agent-server that can be executed and a tutorial to run it. On the other hand, Fig. 2 shows the SPARQL Endpoints dbpedia.org, freebase.org, geonames.org and yago.org, AgentA, AgentB, AgentC and AgentD installed on each server. Each of these agents has implemented a NautiLOD Engine[1,2], which allows to process NautiLOD expressions and delegating tasks if it were necessary.



Expression NautiLOD

Q1: <dbpo:homeTown>
Q2: [ASK ?person <rdf:type> <dbpo:Person>.
?person <rdf:type> <dbpo:MusicalArtist>]
Q3: <dbpo:birthPlace>
Q4: [ASK ?town <dbpo:populationTotal> ?pop.
FILTER(?pop <15000)]
Q5:<owl:sameAs>*

Fig. 2. Architecture for Distributed NautiLOD

3 NautiLOD Distributed Execution

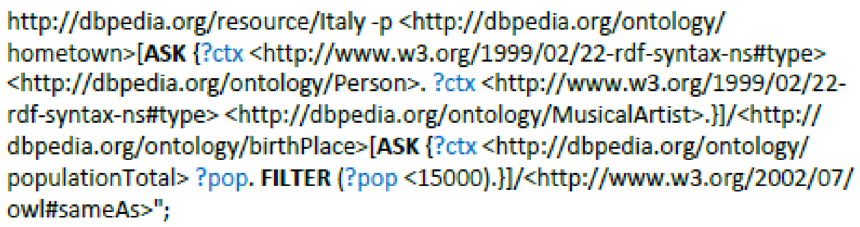

```
http://dbpedia.org/resource/Italy -p <http://dbpedia.org/ontology/
hometown>[ASK {?ctx <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://dbpedia.org/ontology/Person>. ?ctx <http://www.w3.org/1999/02/22-
rdf-syntax-ns#type> <http://dbpedia.org/ontology/MusicalArtist>.}]/<http://
dbpedia.org/ontology/birthPlace>[ASK {?ctx <http://dbpedia.org/ontology/
populationTotal> ?pop. FILTER (?pop <15000).}]/<http://www.w3.org/2002/07/
owl#sameAs>";
```

Fig. 3. NautiLOD language expression with actions.

In http://agentserver.herokuapp.com/nautilodrun.do/ can be found an implementation of Distributed NautiLOD and the following query was tested: *"Starting from DBpedia, find cities with less than 15000 persons, along with their aliases, in which musicians, currently living in Italy, were born"*. Fig. 3 shows the NautiLOD language expression. This expression is an RDF file with the Test ASK interlaced and a FILTER which allows evaluating triples that meet the established pattern. In addition, the expression incorporates actions **putTo(.)** and **exec(.)** which will be interpreted by the players involved, these actions are defined in the specification language for delegating tasks to the environment of the Web[6]. The **putTo(.)** action indicates that the result will be delivered to *agent3@yagos.org* agent after executing the **exec(.)** action.

In what follows, we describe the execution of the Distributed NautiLOD expression using agents:

- The AgentA represents *agent1@dbpedia.org* starts the processing of the NautiLOD expression and it obtains the description of Italy D(dbp:Italy) and looks for URIs having dbp:hometown as a predicate and getting as result those that satisfy this pattern.
- There are several URIs belonging to other Endpoints, but according to the initial expression, our interest is only URIs belonging to the Endpoint geonames.org. The AgentA communicates with the agent AgentB represents *agent2@geonames.org* to send the NautiLOD expression by a message expressed as *msg(AgentA, AgentB, REQUEST(PutTo(AgentC, Exec(Expr, Mdata)))* where AgentC represents agent3@yago.org and Expr represents the NautiLOD expression, and Mdata represents the metadata necessary for the execution of the task. When the request reaches the agent *agent2@geonames.org*, it evaluates the new NautiLOD expression with a reasoning similar to the one of the initial agent and sent by the agent *agent1@dbpedia.org*.
- The AgentB has to check on D(geo: Solarolo) if the query can be satisfied, that is, whether this city has less than 15K habitants The AgentB at geonames.org contacts directly the AgentC to send the result (i.e., the URI geo:Solarolo) by type messages ACL expressed as *msg(AgentB, AgentC, REQUEST(Result(Ri)))*.
- Optionally, the AgentC notifies the result of the task direct to the email of the requesting user.
- The end result of the task execution is represented by Fig. 9, where it can be seen how decreases the workload on each endpoint as each of them gets less and less URI that meet the specified pattern until the URIs obtained in the server http://yagos.org/ give the final result.

4 Preliminary Results

When a task is started the platform generates an identifier, this is to track the same on the other Endpoints. Fig. 4 represents the interaction of the participating agents agent1@dbpedias.org, agent2@geonames.org and agent3@yagos.org and the following process takes place.

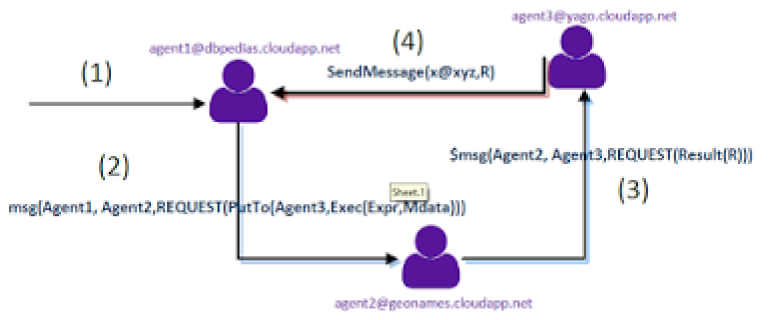

agent3@yago.cloudapp.net
agent1@dbpedias.cloudapp.net
(4)
(1)
SendMessage(x@xyz,R)
(2)
$msg(Agent2, Agent3,REQUEST(Result(R)))
Sheet.1
(3)
msg(Agent1, Agent2,REQUEST(PutTo(Agent3,Exec(Expr,Mdata)))
agent2@geonames.cloudapp.net

Fig. 4. Interaction between distributed agents

1. agent1@dbpedias.org receives the task represented by the expression NautiLOD which includes the actions to be performed the same as shown in Fig. 3. These actions are **PutTo** and **Exec** where the agent agent1@dbpedias.org must execute to resolve the task. The first URIs obtained from the dbpedia.org Endpoint are presented through Fig. 5 which represents all triples that meet the *dbp:hometown* predicate.

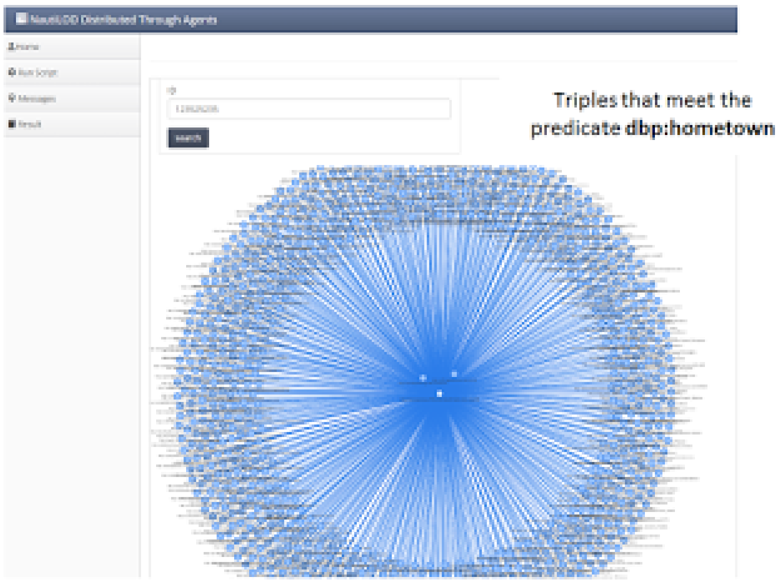

NautiLOD Distributed Through Agents
Home
Run Script
Messages
Result
search
Triples that meet the predicate dbp:hometown

Fig. 5. Result obtained in the Endpoint dbpedia.org

2. The result obtained by agent1@dbpedias.org will check if there are URIs that belong to the geonames.org Endpoint. If in the case of these URI's the agent agent1@dbpedias.org delegates the task to the agent agent2@geonames.org using the FIPA ACL message shown in Fig. 8.

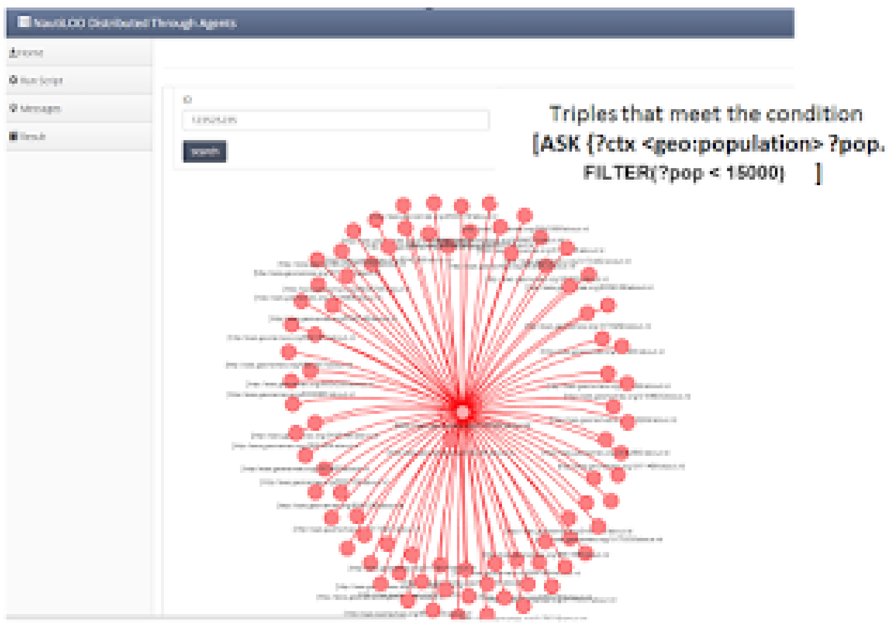

NautiLOD Distributed Through Agents
Home
Run Script
Messages
Result
Search
Triples that meet the condition
[ASK {?ctx <geo:population> ?pop.
FILTER(?pop < 15000)]

Fig. 6. Result obtained in the Endpoint geonames.org

3. The agent agent2@geonames.org evaluates the NautiLOD expressions with a friction similar to the agent agent1@dbpedias.org. The URIs obtained from the Endpoint geonames.org is presented by Fig. 6 which represents all triples that meet the condition **[ASK?ctx<geo:population>?pop:FILTER(?pop>10000):].**

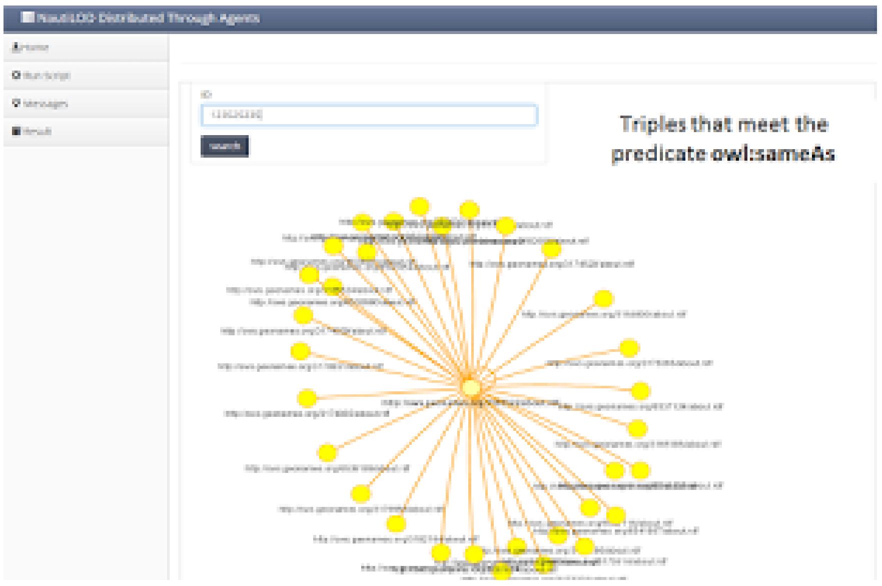
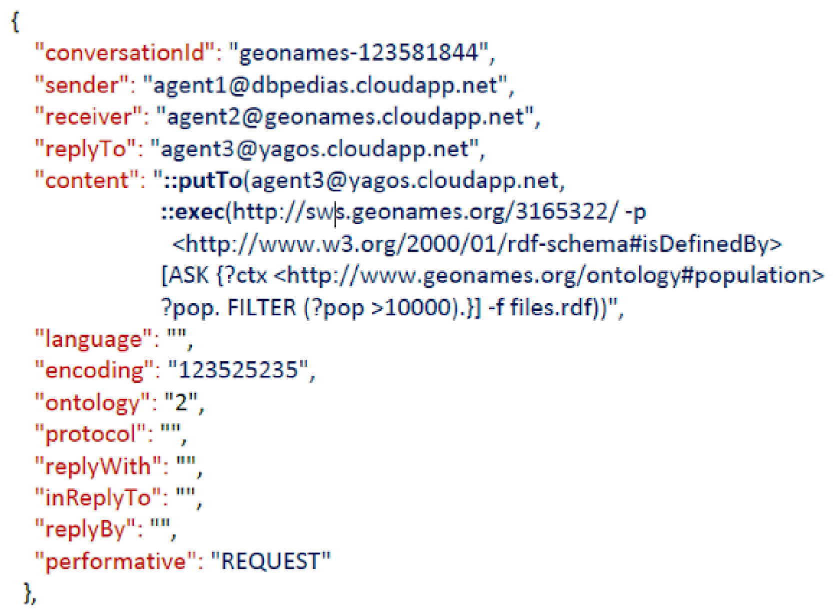

NautiLOD Distributed Through Agents
Home
Run Script
Messages
Result
ID
Search
Triples that meet the predicate owl:sameAs

Fig. 7. Result obtained in the Endpoint yago.org

4. agent2@geonames.org delivers those URIs that meet the specified pattern to the agent agent3@yagos.org using the ACL message represented in Fig. 8. The final result of the task is represented by Fig. 9.
5. The agent agent3@yagos.org receives all the results and performs the writing in its dataset to create the notification of the result to the user later.



```
{
  "conversationId": "geonames-123581844",
  "sender": "agent1@dbpedias.cloudapp.net",
  "receiver": "agent2@geonames.cloudapp.net",
  "replyTo": "agent3@yagos.cloudapp.net",
  "content": "::putTo(agent3@yagos.cloudapp.net,
              ::exec(http://sws.geonames.org/3165322/ -p
               <http://www.w3.org/2000/01/rdf-schema#isDefinedBy>
              [ASK {?ctx <http://www.geonames.org/ontology#population>
              ?pop. FILTER (?pop >10000).}] -f files.rdf))",
  "language": "",
  "encoding": "123525235",
  "ontology": "2",
  "protocol": "",
  "replyWith": "",
  "inReplyTo": "",
  "replyBy": "",
  "performative": "REQUEST"
 },
```

Fig. 8. ACL message

6. Optionally, agent3@yagos.org reported directly to the email, the result obtained.

The final result of the execution of the task is represented by Fig. 9, where it is possible to observe how the workload in each Endpoint is decreasing, obtaining less and less URIs that fulfill the specified pattern, leaving the result of the URIs Obtained in the Server of http://yagos.org.

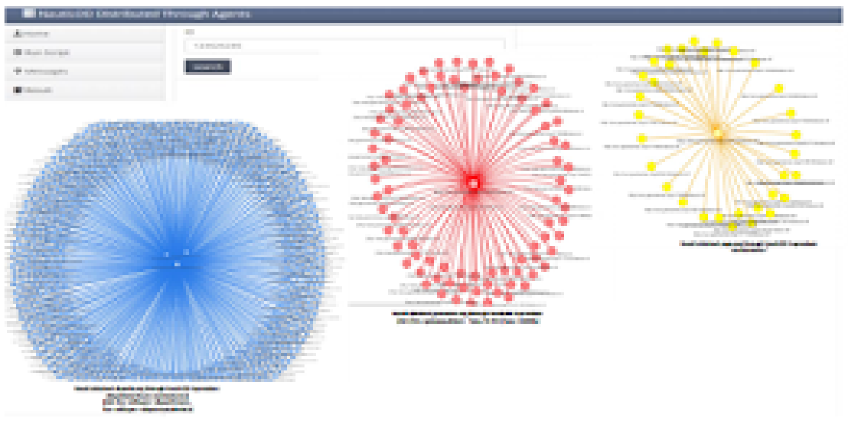


Fig. 9. Final result NautiLOD Distributed

5 Conclusions

At first sight, it may seem this as a simple exercise or example. But, (1) developing this lightweight agent infrastructure (with a basic communication language) that follows basic standards in the Agent field (and thus, able to be further extended); (2) that uses Web protocols (particularly HTTP) and thus scalable; (3) to have been able to implement over it a complex navigational language like NautiLOD; has shown that to integrate these tasks is by no means a trivial development. For this project, the homogeneity of data in the Linked Data cloud was proved to be crucial (and probably one of the most difficult challenges to scale this case to the whole Liked Open Data cloud).

6 Video of Execution

References

1. Valeria Fionda, Claudio Gutierrez, and Giuseppe Pirró. Semantic navigation on the web of data: specification of routes, web fragments and actions. In Proceedings of the 21st international conference on World Wide Web, pages 281–290. ACM, 2012.
2. Valeria Fionda, Giuseppe Pirrò, and Claudio Gutierrez. Nautilod: A formal language for the web of data graph. ACM Transactions on the Web (TWEB), 9(1):5, 2015
3. Valeria Fionda, Claudio Gutierrez, and Giuseppe Pirro. The swget portal: Navigating and acting on the web of linked data. Web Semantics: Science, Services and Agents on the World Wide Web, 26:29–35, 2014
4. Valeria Fionda, Claudio Gutierrez, and Giuseppe Pirró. Semantically-driven recursive navigation and retrieval of data sources in the web of data, 2011.

5. Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. Developing multi-agent systems with JADE, volume 7. John Wiley & Sons, 2007
6. Teofilo Chambilla and Claudio Gutierrez. The notion delegation of tasks in Linked Data through agents.Proceedings of the Alberto Mendelzon Workshop (AMW), Monte Video, Uruguay, Jun 5-9, 2017
7. WOOLDRIDGE, Michael. An introduction to multiagent systems. John Wiley & Sons, 2009.