The notion delegation of tasks in Linked Data through agents

Teófilo Chambilla[1] and Claudio Gutierrez[2]

[1] University of Technology and Engineering, tchambilla@utec.edu.pe,
[2] DCC Universidad of Chile and CIWS cgutierr@dcc.uchile.cl

Abstract. In this exploratory research we study the computational abilities of servers in the Linked Data network through agents that cooperatively evaluate tasks via delegation. We show the feasibility of developing lightweight systems of agents over the current Web infrastructure. We design a simple delegation language, develop a lightweight software (agent) to be installed over Web servers, and implement a study case over actual Web data to show the viability of the proposal.

1 Introduction

In the Web environment, implementing the notion of delegation presents manifold challenges. In this research we aim to show the feasibility of implementing the notion of delegation over current Web protocols and languages by means of simple reactive agent architectures.[3] We present an exploratory study towards leveraging the computational abilities of current servers in the network through agents (using the notion of Multi-Agent Systems (MAS) [14]) that cooperatively evaluate and perform tasks in Linked Data via delegation. We base our approach in the work of Castelfranchi & Falcone [3].

A crucial aspect of our work is the need to develop agents that work over today's Web protocols. There is work in this direction. Communication agents using the methods GET, POST and PUT of the HTTP communication protocol has been recently studied by Abdelkader and Bergeret [13]. There, REST-agents are presented, based on the combination of MAS framework concepts and the principles of REST defined by Roy Fielding [7]. Also, Althagafi [1] proposes a model of agents who follow the REST principles specifying the terms of the agents by *Communicative Acts* of the FIPA architecture and the methods of the HTTP communication protocol. Our approach is based on these works.

Our approach needs a specification language to code the different commands involved in the delegation process. Our design of it is inspired in the work of Doherty et al. [6]. They introduced a task specification language called *Task Specification Tree*, which helps representing the delegation of tasks and that has properties required for troubleshooting joint initiatives and adjustable autonomy for distributed environments. Our task specification language is influenced by this proposal.

[3] This paper is a short report on the Master Thesis of the first author. The spanish version is available at http://repositorio.uchile.cl/handle/2250/139150

2 A Formal Specification of Delegation on the Web

We formulate the delegation mechanism and identify the main characteristics that may occur during the delegation in MAS in the environment of the Web. By way of motivation, consider an organized group of Web servers A, B, C, D, E and in each of them agents $Agent_A$, $Agent_B$, $Agent_C$, $Agent_D$ and $Agent_E$ respectively installed. A possible scenario of interaction of these agents is that $Agent_A$ delegates a tasks to $Agent_B$ and $Agent_C$, and both report the task performed to $Agent_E$, who centralizes and processes the results received from them, to finally deliver the result to $Agent_D$, who in turn can deliver the result to $Agent_A$ or alternatively notify by a messaging system to the original user who originated the task.

A prerequisite for the proper functioning of the scenario described would be the existence of a software infrastructure of MAS so that agents can interact with each other collaboratively. On the Web, a platform environment based entirely on Web protocols allowing robust, secure and reliable interaction is required. This platform should include at least the following capabilities: (1) The ability to process the delegated task; (2) to subdelegate tasks to other actors; (3) to combine the results received; (4) to reason about the results and determine if the result is still necessary to delegate to another agent or the process is terminated; (5) to put the results in the specified place.

For allowing the agents to develop their capacities it is needed a high level of communication infrastructure between platforms. The *Agent Communication Language* (ACL) [12] communication language established by FIPA would be the most appropriate for it. On the other hand, an ideal platform for MAS would be JAVA Agent DEvelopment Framework because it has as characteristic a good communication infrastructure. However, although it supports HTTP protocol communications, is not completely oriented to the Web. For this reason, we develop and used platform called *Agent Server* [4], which is based entirely on the Web and developed with features of a REST API. *Agent Server* 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 and managed by the methods of the HTTP communication protocol. Likewise Agent Manager incorporates the module responsible for the management of the platform and agents.

The core of the MAS is the notion of action, which codes the basic capabilities of the system. In the following we introduce the miminal actions that agents should have to implement the delegation process:

- **Exec**$(Expr, Mdata)$ The agent that has this action implemented should have the ability to run a task. This action is composed of $Expr$ that represents a particular expression (for example, an SPARQL expression, an SQL expression), and $Mdata$ that is the metadata that is required for the implementation of the action (for example it can be either a constraint).
- **PutTo**$(Agents, R)$ The agent that has this action implemented has the ability to make and/or deliver a result obtained after performing an action

Exec(·) to an agent that can be specified initially, or could be the one that initiated a task or a third party that is on a different platform but within the same domain. In this action, *Agents* represents the agent receiving the result and R represents the result obtained.

- **Join**$(R_i, ..., R_n)$ The agent that has this action implemented has the ability to unite the results received and integrate them into one. This actions parameters $R_i, ..., R_n$ represent the results received.

The process of delegation requires agents that might specify arguments and communicate data in a structured manner. The parameters can be about the nature of information search, basic negotiation, or simply express tasks that the agent must perform. The degree of allocation of those parameters will reflect the degree of delegation. We use the notion of *performatives* defined in the *Speech Acts Theory* presented by Austin [2]. In the ACL the *performatives* are denoted as *Communicative Acts* and is expressed as a query, recommendation or call. For our current model we only consider the *performatives* QUERY_IF, REQUEST and INFORM. The choice of these is due to the simplicity of their implementation.

In order to define a formal semantics, we start from the Communicative Act *S-Delegate()*, described by Doherty et al. [5]. Their setting is oriented towards completely reactive agents, that is, agents with very simple reasoning capabilities and under the restrictions of the HTTP communication protocol. The following expression represents represents Communicative Act of the FIPA Architecture:

$$\texttt{S-Delegate}(Agent_A, Agent_B, \texttt{performative}(\tau, Mdata)).$$

This means that the $Agent_A$ delegates the task τ to $Agent_B$, using the FIPA *performatives*. This means that the following conditions must be fulfilled: All agents have the same capabilities, that is, the same actions are implemented in all agents; All agents exhibit the list of actions implemented; The agents belonging to the same domain know of the existence of other agents and therefore know their location; At the same domain all agents have the necessary permissions to perform an action.

The actual behavior of MAS relies in a relevant manner in the specifation of the interaction protocol between participants to ensure interoperability and safety of participants. The protocol describes the interaction between $Agent_A$ and $Agent_B$ and is developed in the following two phases: *Phase 1.* The $Agent_A$ checks the list of actions that are implemented in $Agent_B$. To do this, the *performatives* QUERY_IF (to query the actions to develop) and INFORM (for the result of the query) are used; *Phase 2.* If the received result satisfies the list of actions to be developed, $Agent_A$ plays the role of the delegator agent that delegates the task τ and $Agent_B$ plays the role of contractor agent. To do this, the performatives REQUEST and INFORM are used. By means of REQUEST the task τ is sent to the contractor agent $Agent_B$, who is committed to execute the task. Once it finishes the execution, sends a conclusion message by means of INFORM to $Agent_A$, in order that the delegator does not wait indefinitely.

3 NautiLOD Distributed Execution

In the literature there are different approaches to address the navigation on the Linked Open Data (LOD). Among them is the language NautiLOD [9, 11], 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) this is implemented by SWPORTAL [10] and SWGET [8]. Both SWPORTAL and SWGET are currently implemented in a centralized form, depending to process a NautiLOD expression on a central node that makes the succesive requests to different Endpoints using only the GET method of the HTTP protocol.

In our case study we develop a fully distributed version using agents. For this, four servers have been configured in the Microsoft Azure platform in which we replicated the SPARQL Endpoints *dbpedia.org*, *freebase.org*, *geonames.org* and *yago.org* respectively and agents A, B, C and D installed on each server. Each of these agents has implemented a *NautiLOD Engine*, which allows to process NautiLOD expressions and delegating tasks if it were necessary.

To illustrate, let us show how the following query, described in the literature [9] performs: *"Starting from DBpedia, find cities with fewer than 15000 persons, along with their aliases, in which musicians,currently living in Italy, were born"*.

This expression becomes an ASK query interlaced with a FILTER which allows evaluating triples that meet the pattern established. The agent expression takes the form `::putTo( Agent`$_C$`, ::exec( EXP ))` which will be interpreted by the players involved. The `putTo(·)` action indicates that the result will be delivered to $Agent_C$ after executing the `exec( EXP )` action where `EXP` is the NautiLOD expresssion.

The execution of the NautiLOD expression uses delegation as follows: (1) $Agent_A$ starts the processing of the NautiLOD expression and it obtains the first description (for example, *D(dbp:Rome)*) and looks for URIs having *dbp:hometown* as a predicate and getting as result those that satisfy this pattern; (2) There are several URIs belonging to other Endpoints, but according to the initial expression our interest are only URIs belonging to the Endpoint *geonames.org*. $Agent_A$ communicates with $Agent_B$ to send the NautiLOD expression by a message ACL expressed as Communicative Act of type $msg(Agent_A, Agent_B, REQUEST(PutTo(Agent_C, Exec(Expr, Mdata)))$ where $Agent_C$ represents the agent receiver and $Expr$ represents the new NautiLOD expression, and $Mdata$ represents the metadata. When the request reaches to $Agent_B$, it evaluates the new NautiLOD expression with a reasoning similar to the one of the initial agent; (3) $Agent_B$ has to check if the query can be satisfied, that is, whether this city has less than 15K habitants. $Agent_B$ at geonames.org contacts directly the $Agent_C$ to send the result (for example, the URI *geo:Solarolo*) by type messages ACL $msg(Agent_B, Agent_C, REQUEST(Result(R_i)))$; (4) Optionally, $Agent_C$ notifies the result of the task direct to the email of the requesting user.

4 Conclusions

The design of a specification language for the delegation of tasks in the environment of the Web (really in Linked Open Data) poses several challenges. The work presented here shows the feasibility of implementing a lightweight system of agents using current Web infrastructure (communication protocols, LOD system of distributed data, and available proposals of navigational languages).

At first sight it may seem this as a simple excercise or example. But it development needs: (1) developing a lightweight agent infrastructure (with a basic communication language) that follows basic standards in the Agent field (and thus, able to be further extended); (2) using only Web protocols (particularly HTTP) and to make it scalable; (3) to have power to be able to implement over it a complex navigational language like NautiLOD. To integrate these tasks in a modula fashion is by no means a trivial endeavor.

References

1. A. H. H. Althagafi. Designing a framework for restful multi-agent systems. Master's thesis, Department of Computer Science, University of Saskatchewan, 2012.
2. J. L. Austin. *How to do things with words*, volume 367. Oxford Univ. Press, 1975.
3. C. Castelfranchi and R. Falcone. Towards a theory of delegation for agent-based systems. *Robotics and Autonomous Systems*, 24(3):141–157, 1998.
4. T. Chambilla. Agent Server Platform. `https://github.com/tchambil/agent-server`, 2015. [Online; accessed 01-March-2017].
5. P. Doherty, F. Heintz, and D. Landén. A delegation-based architecture for collaborative robotics. In *Agent-Oriented Software Engineering XI*, pages 205–247. Springer, 2011.
6. P. Doherty, F. Heintz, and D. Landén. A distributed task specification language for mixed-initiative delegation. In *Principles and Practice of Multi-Agent Systems*, pages 42–57. Springer, 2012.
7. R. Fielding. Representational state transfer. *Architectural Styles and the Design of Netowork-based Software Architecture*, pages 76–85, 2000.
8. V. Fionda, C. Gutierrez, and G. Pirró. Semantically-driven recursive navigation and retrieval of data sources in the web of data, 2011.
9. V. Fionda, C. Gutierrez, and G. 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.
10. V. Fionda, C. Gutierrez, and G. 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.
11. V. Fionda, G. Pirrò, and C. Gutierrez. N auti lod: A formal language for the web of data graph. *ACM Transactions on the Web (TWEB)*, 9(1):5, 2015.
12. T. FIPA. Fipa communicative act library specification. *Foundation for Intelligent Physical Agents, http://www. fipa. org/specs/fipa00037/SC00037J. html (30.6. 2004)*, 2008.
13. A. Gouaïch and M. Bergeret. Rest-a: An agent virtual machine based on rest framework. In *Advances in Practical Applications of Agents and Multiagent Systems*, pages 103–112. Springer, 2010.
14. M. Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.