NOWHERE - An Open Service Architecture to support Agents and Services within the Semantic Web

Nicola Dragoni, Mauro Gaspari, and Davide Guidi

Dipartimento di Scienze dell'Informazione, University of Bologna, ITALY
{gaspari, dragoni, dguidi}@cs.unibo.it

Extended Abstract

Agents are one of the main building blocks of the World Wide Web under the Semantic Web activity [1]. The Web is evolving toward an open, service-oriented architecture, which is a software infrastructure that makes an open set of information services available to users and agents. The role of agents is to retrieve, execute and compose available services providing more sophisticated instances of them. Knowledge is playing an increasingly important role in this scenario, services are being extended with semantic information [11] and agents will provide complex problem solving capabilities to perform their tasks.

Although there are many small-scale examples of implemented systems which use formalised knowledge to achieve intelligent behaviours, for example personal assistants which collect and organise information, there is still a major gap between these isolated systems and the vision of the Semantic Web.

Agents have been recognised as one of the main technologies to fill this gap [10], but their role in the emerging Semantic Web infrastructure is still not completely depicted. For example, their relationship with more standard components such as Web Servers and clients, is still not clear. Most of the examples of agents in the Web are **User agents** which provide intelligent support and advanced services to users. However, whenever an agent provides a set of complex problem solving capabilities, it becomes reasonable to reuse its skill in realizing other intelligent applications. This suggests another possible role for agents: to enhance the functionality of servers. **Worker agents**, providing complex problem solving capabilities with respect to a given application domain, can be published and shared on the Web, for example by means of a set of well defined Web Services and an associated ontology. However, there is still not a widely accepted architecture for integrating agents in a distributed reasoning infrastructure on the Web.

In this paper we highlight the features of our NOWHERE architecture, an Open Service Architecture (OSA) that supports agents and services within the Semantic Web, which is described in detail in [6,7]. A general view of our architecture is depicted in Figure 1.

Agents can be both Worker agents, which can be published in Web agent servers, or User agents which, if necessary, can be downloaded and activated in mobile devices. These agents can be realized with any programming language

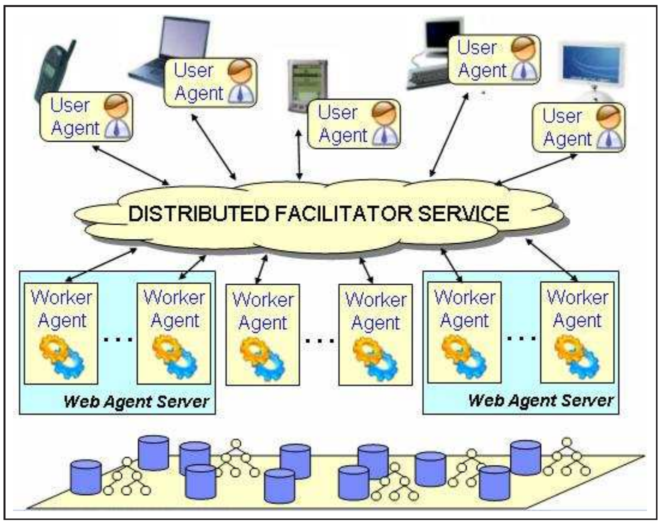

Fig. 1. Architecture of the agent-based OSA.

or traditional AI technologies provided that they are able to access the RDF-based triple-space. Agents are connected through a **Distributed Facilitator Service**. In the following we summarise the main features of our architecture.

Dynamic and Open Architecture. In the research on Multi-Agent Systems there is an increasing emphasis on the open-ended nature of agent systems, which refers to the feature of allowing the dynamic integration of new agents into an existing agent system. This feature becomes particularly relevant when agents are developed on the Web where they are usually implemented by different people at different times. Our OSA allows new agents to dynamically connect themselves to the system, providing new capabilities to other agents.

High Level Inter-Agent Communication. Agents access services and communicate with each other using an advanced high level Agent Communication Language (𝔽𝕋-𝔸ℂ𝕃) [3,5]. 𝔽𝕋-𝔸ℂ𝕃 provides a small set of fault-tolerant conversation performatives and supports a fault-tolerant *anonymous interaction protocol* based on one-to-many requests for knowledge. Moreover 𝔽𝕋-𝔸ℂ𝕃 has been designed for *open* architectures and deals with a *dynamic* set of agents [4].

Integration of Agents and Web Services. Agents can be published as Web Services and existing Web Services can be registered or agentified enhancing agents' ability to discover and invoke them with the fault-tolerant ACL. This integration can be configured dynamically realizing an Open Service Architecture. New Web Agent Servers can be added dynamically to provide one or more Web Services. This set of Web Services can dynamically change because of the creation of new services on the Web Agent Server or the modification of existing ones.

Support for Interoperability. In our architecture agents can be realized in any programming language including AI languages or knowledge representation languages, provided that they react to a well defined protocol based on the standard primitives of $\mathbb{FT}$-$\mathbb{ACL}$. Although all the emerging standards for the Semantic Web use formalisms based on XML, most of AI systems are still being developed using specific AI technologies and languages which usually are not compliant with Web standards, but provide powerful engines and a rich set of libraries. From a practical point of view it is not feasible to translate all these technologies in XML based formalism or to commit to a single programming language. Thus, enabling the integration of agents written in different programming languages is essential to a large scale exploitation of Knowledge-Level agents on the Web.

Sound Failure Model. To support distributed reasoning protocols which function in presence of failure we adopt a well-known failure model. We assume agents are subject to possible failures. Following a well-known classification of process failures in distributed systems [12], we say that an agent is *faulty* in an execution if its behaviour deviates from that prescribed by the algorithm it is running; otherwise, it is *correct*. The failure model we consider is *crash* failures of agents in a *fully asynchronous* Multi-Agent System: a faulty agent has *crashed* if it stops prematurely and does nothing from that point on. Before stopping, however, it behaves correctly. Note that considering only crash failures is a common fault assumption in distributed systems, since several mechanisms can be used to detect more severe failures and to force a crash in case of detection. $\mathbb{FT}$-$\mathbb{ACL}$ primitives are designed to react to agents' crashes. In this way several concurrent and fault tolerant properties of the ACL can be proved [2,5].

Knowledge-Level Requirements. One of the main features of our OSA is its Knowledge-Level characterization including support for Knowledge-Level agents and Knowledge-Level communication using $\mathbb{FT}$-$\mathbb{ACL}$. We assume *Knowledge-Level* agents, that is, agents concerned with the use, request and supply of knowledge which do not deal with symbol level issues. In [8] requirements for Knowledge-Level communication are postulated which need a careful analysis of the ACL and the underlying agent architecture in order to ensure Knowledge-Level behaviour. We list again these *Knowledge-Level Programming Requirements* below extended to deal with crashes of agents (condition (4)).

(1) The programmer should not have to handle physical addresses of agents explicitly.
(2) The programmer should not have to handle communication faults explicitly.
(3) The programmer should not have to handle starvation issues explicitly. A situation of starvation arises when an agent's primitive never gets executed despite being enabled.
(4) The programmer should not have to handle *communication deadlocks* explicitly. A communication deadlock situation occurs when two agents try to communicate, but they do not succeed; for instance because they mutually wait for each other to answer a query [13] or because an agent endlessly waits for a reply of a crashed agent.

FT-ACL supports these Knowledge-Level requirements [2,3]. Note that the introduction of these requirements becomes particularly relevant in open agents' architectures which use ACLs for inter-agent communication. In fact, if an ACL supports these Knowledge-Level requirements several properties can be inferred in the resulting agent system, for example:

– **Deadlock Avoidance**: new agents can be freely added to the system without introducing communication deadlocks. This is guaranteed by requirement (4).
– **Firewalls Independence**: most Internet subnets employ firewalls to protect their hosts from Internet invaders and attacks. Thus agents developed in different networks often have to go through firewalls to communicate. Requirement (1) states the programmer of agents should not have to handle physical addresses of agents explicitly. Thus agents should be reachable using their logical names which should not depend of the physical information that is usually analyzed by packet firewalls such as port numbers.

Ontology Driven Communication: We assume two agents must share a **service ontology** to communicate. According to Gruber [9], an ontology is a formalization of a shared conceptualization. In our architecture a service ontology describes all the services of a given application domain and the related terminology. As soon as an agent is created it must commit to a specific service ontology to take part to a conversation. Two agents can interact together only after a prior agreement upon a shared service ontology. In fact in our vision ontologies for agents are similar to languages for human beings. Concerning the description of individual services our architecture supports both standard formalisms such as WSDL and more powerful formalisms to realize Semantic Web Services.

In summary, our approach demonstrates that different issues, such as high-level inter-agent communication and fault tolerance, can be successfully integrated in an open and dynamic agent-based OSA which provides Web Services maintaining a clean design of the architecture and a Knowledge-Level characterization.

References

1. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284, 2001.
2. N. Dragoni and M. Gaspari. An Object Based Algebra for Specifying A Fault Tolerant Software Architecture. *Journal of Logic and Algebraic Programming*, 63(2):271–297, 2005.
3. N. Dragoni and M. Gaspari. Fault Tolerant Knowledge Level Communication in Open Asynchronous Multi-Agent Systems. Technical Report UBLCS-2005-10, Department of Computer Science, University of Bologna (ITALY), April 2005.
4. N. Dragoni, M. Gaspari, and D. Guidi. A Fault Tolerant Agent Communication Language for Supporting Web Agent Interaction. In *Agent Communication - International Workshop on Agent Communication, AC 2005*, To appear in Lecture Notes in Computer Science. Springer Verlag.

5. N. Dragoni, M. Gaspari, and D. Guidi. An ACL for Specifying Fault-Tolerant Protocols. In *Proceedings of AIIA Conference*, Lecture Notes in Computer Science, pages 237–248, Milan, ITALY, 2005. Springer Verlag.
6. N. Dragoni, M. Gaspari, and D. Guidi. Integrating Knowledge-Level Agents in the (Semantic) Web: an Agent-based Open Service Architecture. In *Proceedings of the 18th International FLAIRS Conference*, Clearwater Beach, Florida, USA, 2005. AAAI Press.
7. N. Dragoni, M. Gaspari, and D. Guidi. An Infrastructure to Support Cooperation of Knowledge-Level Agents on the Semantic Grid. To appear in International Journal of Applied Intelligence (Special Issue on Agent-based Grid Computing), 2006.
8. M. Gaspari. Concurrency and Knowledge-Level Communication in Agent Languages. *Artificial Intelligence*, 105(1-2):1–45, 1998.
9. T. Gruber. A Translation Approach to Portable Ontologies. *Knowledge Acquisition*, 5(2):199–220, 1993.
10. J. Hendler. Agents and the Semantic Web. *IEEE Intelligent Systems*, 16(2), 2001.
11. S. McIlraith and D. Martin. Bringing Semantics to Web Services. *IEEE Intelligent Systems*, 18(1):90–93, 2003.
12. S. Mullender. *Distributed Systems*. Addison Wesley, 1993.
13. M. Singhal. Deadlock Detection in Distributed Systems. *IEEE Computer*, 22(11):37–48, 1989.