A CONTEXT-AWARE SEMANTIC WEB SERVICE EXECUTION AGENT

António Luís Lopes and Luís Miguel Botelho
"We, the Body, and the Mind" Research Lab of ADETTI-ISCTE, Avenida das Forças Armadas
Edifício ISCTE, 1600-082, Lisboa, Portugal

Keywords: Service Execution, Semantic Web, Context-awareness.

Abstract: This paper presents the research on agent technology development for context-aware execution of semantic web services, more specifically, the development of SEA (Service Execution Agent), a semantic web services execution agent that uses context information to adapt the execution process to a specific situation, thus improving its effectiveness and providing a faster and better service. Preliminary results show that the introduction of context information and context-aware capabilities in a service execution environment can speed up the execution process, in spite of the overhead that it is introduced by the agents' communication and processing of context information. The developed service execution agent uses standards such as OWL-S service descriptions and WSDL grounding information. Also, an *AgentGrounding* definition has been proposed to enable the execution of semantic web services provided by agents.

1 INTRODUCTION

Standard initiatives such as OWL-S and WSDL (Martin, 2004) enable the automation of discovery, composition and execution of semantic web services, i.e. they create a Semantic Web, such that computer programs or agents can implement an open, reliable, large-scale interoperation of Web Services. In this paper we describe part of our research done on agent technology development for context-aware execution of semantic web services.

We have decided to adopt the agent paradigm, creating the SEA (Service Execution Agent) agent, because we intend to integrate this work in open societies of agents, enabling these to execute semantic web services. (Paolucci, 2004) used the same approach in the Web Services infrastructure because of its capability to perform a range of coordination activities and *anonymising* between requesters and providers. In our research, the use of context information helps improve the execution process, by adding valuable situation-aware information that will contribute to its effectiveness.

The major contributions of the present work to advance the state-of-the-art are: i) the development of a broker agent capable of executing received OWL-S/WSDL service descriptions; ii) the inclusion of context-awareness into semantic web services execution. The rest of this paper is organized as follows: section 2 gives a brief overview on the use of context; section 3 describes the created broker agent; section 4 presents some preliminary results of the evaluation; section 5 concludes and presents guidelines for future work.

2 CONTEXT-AWARENESS

Context-aware computing is a computing paradigm in which applications can discover and take advantage of context information to improve their behaviour in terms of effectiveness as well as performance. As described in (Dey and Abowd, 1999) context is any information that can be used to characterize the situation of an entity. Entities may be persons, places or objects considered relevant to the interaction between a user and an application, including users and applications themselves.

We can enhance this definition by stating that applications are also considered to be entities. SEA bases its activity in the context-aware computing paradigm, where it uses and analyses context information to enhance its service execution process, by adapting it to the specific situation in which the agent and its client are involved, at the time of the execution process. This is done by interacting with a general purpose (i.e., domain independent) context system (Costa and Botelho, 2005) for acquiring context information, subscribing desired context events and providing relevant context information.

Throughout the execution process, SEA provides and acquires context information from and to this context system. For example, SEA provides relevant context information about itself, such as its queue of service execution requests and the average time of service execution. This will allow other entities in the environment to determine the service execution agent with the smallest work-load, and hence the one that offers a faster execution service.

During the execution of a compound service, SEA invokes atomic services from specific service providers (both web services, and service provider agents). SEA also provides valuable information regarding these service providers' availability and average execution time. Other entities can use this information to rate service providers or to simply determine the best service provider to use in a specific situation. Furthermore, SEA uses its own context information (as well as information from other sources and entities in the environment) to adapt the execution process to a specific situation. For instance, when selecting among several providers of some service, SEA will choose the one with better availability (with less history of being offline) and lower average execution time.

In situations such as the one where service providers are unavailable, it is faster to obtain the context information from the context system (as long as service providers can also provide context information about their own availability) than by simply trying to use the services and finding out that they are unavailable after having waited for a connection timeout to occur. If SEA learns that a given service provider is not available it will contact a service discovery agent or a service composition agent requesting that a new service provider is discovered or that the compound service is re-planned. This situation-aware approach using context information on-the-fly helps SEA to provide a value-added execution service.

3 SEA: THE SERVICE EXECUTION AGENT

The Service Execution Agent (SEA) is a broker agent that provides context-aware execution of services in the Semantic Web (whether they are provided by web services or agents). The agent was designed and developed considering the interactions described in section 2. Its internal architecture was clearly designed to enable the agent to receive requests from client agents, acquire and provide relevant context information, interacting with other service coordination agents when necessary and execute remote services.

3.1 Internal Architecture

The developed agent is composed of three components: the Agent Interaction Component (AIC), the Engine Component (EC) and the Service Execution Component (SEC). Figure 1 illustrates the internal architecture of the agent and the interactions that occur between the components.

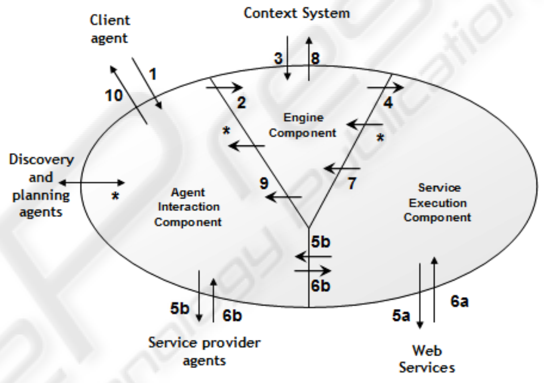

Figure 1: SEA Internal Architecture and Interactions.

The AIC was developed as an extension of the JADE platform (Bellifemine, 1999) and its goal is to provide an interaction framework to FIPA-compliant agents (FIPA, 2002), such as SEA's clients (requesting the execution of specified services – Figure 1, step 1) and service discovery and composition agents (when SEA is requesting the re-discovering and re-planning of specific services – Figure 1, steps *). Among other things, the AIC is responsible for receiving messages, parsing them and processing them into a suitable format for the EC to use it (Figure 1, step 2). The reverse process is also the responsibility of AIC – receiving data from the EC and processing it into the agents' suitable format to be sent as messages (Figure 1, step 9).

The EC is the main component of SEA as it controls the agent's overall activity by pre-processing service execution requests, interacting with the context system and deciding when to interact with other agents (such as service discovery and composition agents). When the EC receives an OWL-S service execution request (Figure 1, step 2), it acquires suitable context information (regarding potential service providers – Figure 1, step 3) and plans the execution process.

If the service providers of a certain atomic service (invoked in the received composed service) are not available, SEA interacts with a service discovery agent (through the AIC – Figure 1, steps *) to discover available providers for the atomic services that are part of the OWL-S compound service. If the service discovery agent cannot find adequate service providers, the EC can interact with a service composition agent asking it to create an OWL-S compound service that produces the same effects as the original service.

After having a service ready for execution, with suitable context information, the EC sends it to the SEC (Figure 1, step 4), for execution. Throughout the execution process, the EC is also responsible for providing context information to the context system, whether it is its own information (such as service execution requests' queue, average time of execution) or other entities' relevant context information (such as availability of providers and average execution time of services).

The SEC was developed as an extension of the OWL-S API (Sirin, 2004) and its goal is to execute semantic web services (Figure 1, steps 5a and 6a) described using OWL-S service description and WSDL grounding information. The extension of the OWL-S API allows for the evaluation of logical expressions in conditioned constructs, such as the *If-then-Else* and *While* constructs, and in the service's pre-conditions and effects. OWL-S API was also extended in order to support the execution of services that are grounded on service provider agents (Figure 1, steps 5b, 6b). This extension is called *AgentGrounding* and it is explained in detail in (Lopes and Botelho, 2005). When the SEC receives a service execution request from the EC, it executes it according to the description of the service's process model. During the execution process, SEC collects relevant context information (such as providers' availability, quality of service and execution times). After execution of the specified service and generation of its results, the SEC sends them to the EC (Figure 1, step 7) for further analysis and post-processing, which includes sending gathered context information to the context system (Figure 1, step 8) and sending the results to the client agent (through the AIC – Figure 1, steps 9, 10).

3.2 Execution Process

OWL-S is an OWL-based ontology used to describe semantic web services. OWL-S Services are described in three parts: a *Profile* (which tells "what the service does"); a *Process Model* (which tells "how the service works"); and a *Grounding* (which tells "how to access the service").

The general approach for the execution of OWL-S services consists of the following sequence of steps: i) validate the service's pre-conditions, whereas the execution process continues only if all pre-conditions are true; ii) decompose the compound service into individual atomic services, which in turn are executed by evoking their corresponding service providers (described in the grounding section of the service); iii) validate the service's described effects by comparing them with the actual effects of the service execution, whereas the execution process is only valid if the service has produced the expected effects; iv) collect the results, if any (the service may be only a "*change-the-world*" kind of service), and send them to the client who requested the execution.

4 EVALUATION

The enhancement provided by the use of context information consists on the adaptation of the execution agent to a specific situation (according to available context information) in a way such that the execution is done in the requested time-frame (by the client agent) and that in case of failure of some of the elements of the compound service, suitable alternatives can be found. Using context information, SEA can determine who the "best" service providers are, by building a sort of "reputation" schema of the available service providers, which then can be used to determine the fastest way to execute a compound service or find alternatives in case of failure.

Even though this approach of service coordination (the combination of service execution with discovery and composition planning) improves the way compound services are executed (by allowing the determination of the best service providers in a specific situation and by providing a failure recovery method), it introduces an "overhead" time that doesn't exist in the actual semantic web services execution environments. This "overhead" time is composed of the procedures that SEA must perform when communicating and managing conversations with other agents (client, service discovery and composition agents), retrieving and processing context information (related to the service providers) and preparing the execution of compound services. It is important to determine how this "overhead" time influences the overall time of execution.

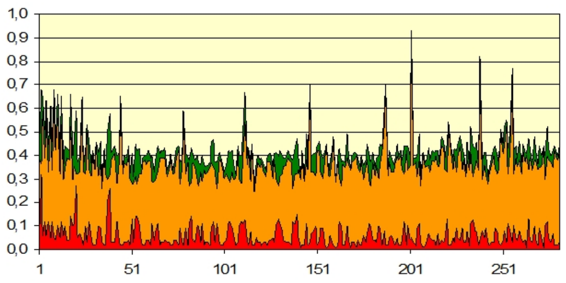

Figure 2: Execution test of a compound service with five atomic services (280 sequential requests measured in seconds).

Figure 2 shows the average behavior of SEA in the sequential execution (280 times) of a compound service (with 5 atomic services), by illustrating the different parts of the execution process. The orange area represents the Normal Execution Time (NET) – this is the time that it takes to execute the web services; the red area represents the Overhead Execution Time (OET) – this is the time that SEA takes to perform the mentioned procedures (conversations' management, acquiring context information and preparing execution); the green area represents the Total Execution Time (TET) – this is the total time of execution: NET + OET.

The average "overhead" execution time is very small (0,06 seconds). This value is hardly noticeable by a user of such a system. Moreover, in some tests the use of this kind of service coordination approach allowed the reduction of the overall execution time due to the fact that SEA always tried to find the fastest service provider (the one with lower history of "down" time, work load and average execution time). This shows that SEA can be used in highly dynamic environments by efficiently distributing the "execution work" to the appropriate service providers, hence providing a faster and more reliable service to a user with time, device and location constraints.

5 CONCLUSIONS

We have presented a framework to enable the execution of semantic web services using a context-aware broker agent. Test results show that the introduction of context-aware capabilities and the interaction within a service coordination infra-structure not only add a very small overhead to the execution process, as it turns SEA into a highly efficient broker agent capable of distributing the execution work among the available service providers, thus providing a more useful and faster service to its clients.

So far, SEA has been tested in an e-commerce book search service environment and the results are promising. The use of SEA in the CASCOM project (Helin, 2005) will allow us to determine its applicability and usability in different scenarios operating in highly dynamic environments.

ACKNOWLEDGEMENTS

This work has been partly supported by the European Commission under the project grant FP6-IST-511632-CASCOM.

REFERENCES

Bellifemine F., Poggi A., Rimassa G., 2001. Developing multi-agent systems with a FIPA-compliant agent framework. Software-Practice and Experience 31 (2): 103–128 Feb 2001

Costa, P., Botelho, L., 2005. Generic Context Acquisition and Management Framework. First European Young Researchers Workshop on Service Oriented Computing. Forthcoming.

Dey, A. K. and Abowd, G. D., 1999. Towards a better understanding of context and context awareness. GVU Technical Report GIT-GVU-99-22, College of Computing, Georgia Institute of Technology.

FIPA Members. 2002. Foundation for Intelligent Physical Agents website. http://www.fipa.org/.

Helin, H., Klusch, M., Lopes, A., Fernandez, A., Schumacher, M., Schuldt, H., Bergenti, F., and Kinnunen, A., 2005. CASCOM: Context-Aware Service Co-ordination in Mobile P2P Environments. Multiagent System Technologies, Lecture Notes in Computer Science, Vol. 3550 / 2005, ISSN: 0302-9743, pp. 242 - 243

Lopes, A., Botelho, L.M., 2005. SEA: a Semantic Web Services Context-aware Execution Agent. AAAI Fall Symposium on Agents and the Semantic Web. Arlington, VA, USA.

Martin, D., Burstein, M., Lassila, O., Paolucci, M., Payne, T., McIlraith. S., 2004. Describing Web Services using OWL-S and WSDL. DARPA Markup Language Program.

Paolucci, M., Soudry, J., Srinivasan, N., Sycara, K., 2004. A Broker for OWL-S Web Services. First International Semantic Web Services Symposium, AAAI Spring Symposium Series.

Sirin, E., (2004). OWL-S API project website. http://www.mindswap.org/2004/owl-s/api/.