Analysis and Design of a SOA-Based Multi-agent Architecture

Dante I. Tapia, Ricardo S. Alonso, Carolina Zato,
Oscar Gil, and Fernando De la Prieta

Abstract. One of the most prevalent approaches among distributed architectures is Multi-Agent Systems. The agents have characteristics such as autonomy, reasoning, reactivity, social abilities and pro-activity which make them appropriate for developing distributed systems based on highly dynamic scenarios. Nevertheless, the development of a multi-agent system can be an extensive and delicate process. During this process, it is convenient to utilize Agent-Oriented Software Engineering (AOSE) tools. Such tools facilitate and improve the engineering process, achieving models that are more detailed and closer to the multi-agent systems implementation. This paper presents the analysis and design of a *Flexible and User Services Oriented Multi-agent Architecture* (FUSION@). This is a new architecture that integrates intelligent agents with a service-oriented approach to facilitate and optimize the development of highly dynamic distributed systems.

Keywords: Distributed Architectures, Multi-Agent Systems, Service-Oriented Architectures, Agent-Oriented Software Engineering, Ontology Design.

1 Introduction

The development of dynamic and distributed software usually requires creating increasingly complex and flexible applications. In some cases, applications require similar functionalities already implemented into other systems which are not always compatible. At this point, developers can face this problem through two options: re-use functionalities already implemented into other systems; or re-deploy the capabilities required. While the first option is more adequate in the long-run, the second one is the most chosen by developers. However, the latter is a poorly scalable and flexible model with reduced response to change, in which applications are designed from the outset as independent software islands.

Dante I. Tapia · Ricardo S. Alonso · Carolina Zato · Oscar Gil · Fernando De la Prieta
Departamento de Informática y Automática, Universidad de Salamanca,
Plaza de la Merced, S/N, 37008, Salamanca, Spain
e-mail: {dantetapia,ralorin,carol_zato,fer,oscar.gil}@usal.es

Y. Demazeau et al. (Eds.): Trends in PAAMS, AISC 71, pp. 183–190.
springerlink.com

For these reasons, it is necessary to develop new functional architectures capable of providing adaptable and compatible frameworks. A functional architecture defines the physical and logical structure of the components that make up a system, as well as the interactions between those components. There are Service-Oriented Architectures (SOA) [1] and agent frameworks and platforms which provide tools for developing distributed systems and Multi-Agent Systems (MAS) [2]. However, these tools do not solve by themselves which the development of current systems needs. Therefore, it is necessary to develop innovative solutions that integrate different approaches to create flexible and adaptable systems.

One of the most prevalent alternatives in distributed architectures is agent and multi-agent systems which can help to distribute resources and reduce the central unit tasks [3]. There are several agent frameworks and platforms, such as OAA [4], RETSINA [5] and JADE [6], which provide a wide range of tools for developing distributed multi-agent systems. An agent-based architecture provides more flexible ways to move functions to where actions are needed, thus obtaining better responses at execution time, autonomy, services continuity, and superior levels of flexibility and scalability than centralized architectures [2]. Additionally, the programming effort is reduced because it is only necessary to specify global objectives so that agents cooperate in solving problems and reaching specific goals, thus giving the systems the ability to generate knowledge and experience. Unfortunately, the difficulty in developing a multi-agent architecture is higher because there are no specialized programming tools to develop agents [7].

Furthermore, the development of a distributed MAS can be an extensive and delicate process. During this process, it is convenient to utilize Agent-Oriented Software Engineering (AOSE) tools. Among this tools, it can be emphasized the Gaia methodology [8] and the SysML modeling language [9]. Such tools facilitate and improve the engineering process, thus achieving models that are more detailed and closer to the MAS implementation. In addition, the use of ontologies provides a powerful tool for knowledge sharing, re-use and interoperability [10].

In the next section, the specific problem description that essentially motivated the development of FUSION@ is presented. Section 3 describes the main characteristics of this architecture and explains some of its components, making an emphasis on the analysis and design process of this development. Finally, section 4 presents the results and conclusions obtained.

2 Problem Description

Distributed architectures like SOA or MAS consider integration and performance aspects that must be taken into account when functionalities are created outside the system. Distributed architectures are aimed at the interoperability between different systems, distribution of resources, and the lack of dependency of programming languages [1]. Functionalities are linked by means of standard communication protocols that must be used by applications in order to share resources in the network [3]. The compatibility and management of messages between functionalities is an important and complex element in any of these approaches.

Among the existing agent frameworks and platforms we have OAA, RETSINA and JADE. Nevertheless, integration is not always achieved because of the incompatibility among them. The integration and interoperability of agents and multi-agent systems with SOA and Web Services approaches has been recently explored [3]. Some developments are centered on communication between these models, while others are centered on the integration of distributed services into the structure of the agents. Bonino da Silva *et al.* (2007) propose merging multi-agent techniques with semantic Web Services to enable dynamic, context-aware service composition. Ricci *et al.* (2007) have developed a Java-based framework to create SOA and Web Services compliant applications, which are modeled as agents.

When designing MAS it is necessary to utilize an AOSE process. There are several AOSE methodologies, languages and tools, such as Gaia, Tropos, MESSAGE, AUML, SysML, etc. Although many of them have important limitations and there are not specialized development tools, the use of AOSE tools vastly facilitates the development of MAS. In the development presented in this paper, it has been chosen the Gaia [8] and the System Modeling Language (SysML) [9] tools, because their integration allows obtaining models that are very closer to the systems implementation in a relatively fast and efficient way. Gaia is a simple and very general methodology focused on the analysis and design of MAS. The Gaia methodology is aimed at analyzing the system and designing its structure from a roles collection called organization. In Gaia the system is not detailed enough to carry out a direct implementation, but the aim is reducing the abstraction level to the point in which it can be applied traditional development techniques. After finalizing the Gaia analysis and design stages it is achieved a too high abstraction level. However, this can be an advantage for developers because there is not a dependency with a particular implementation solution. In the other hand, SysML is a robust language focused on the systems engineering and based on UML (Unified Modeling Language) [9]. Although this language is not considered to be only among the AOSE tools, it provides many characteristics that make it adequate for the design and representation of complex MAS, even taking the place of languages as AUML.

Besides using AOSE tools, it can be very useful for the analysis of systems and architectures to create ontologies that represent the entities involved on them. An ontology can be defined as an explicit specification of conceptualization [10]. This means that an ontology can represent the description of entities and relationships that an agent realizes [13]. There are many languages that allow encoding a certain ontology. The DARPA Agent Markup Language (DAML) was mainly developed for creating machine-readable representations of the Web, thus building what is known as the Semantic Web [14]. DAML syntax is based on the Resource Description Framework (RDF), a W3C recommendation for representing metadata on the web. Furthermore, the RDF Schema (RDFS) is a language created for defining ontological entities by means of RDF. However, as RDFS was not expressive enough for some purposes, the DAML project finally adopted the Ontology Inference Layer (OIL), which provides a fast reasoning support. Nevertheless, OIL has some restrictions that make it not suitable for some applications [10]. This way, DAML+OIL led to the Web Ontology Language (OWL), with several versions available: OWL Lite, OWL DL (Description Logic) and OWL Full.

The analysis and design process of FUSION@ using Gaia and SysML is presented in the following section. In addition, an ontology of the architecture has been built to model the entities implied on FUSION@ and to enhance the possibilities of sharing the knowledge involved on them.

3 Analysis and Design of FUSION@

This section is focused on describing the analysis and design process conducted during the development of FUSION@, an architecture aimed at developing intelligent highly dynamic environments that integrates intelligent agents and a service-oriented philosophy. Thus, the main objective of this paper is to describe this process, not the architecture itself, mainly because the research on FUSION@ is still under development and its description has already been published [15]. The architecture proposes several features capable of being executed in dynamic and distributed environments. These features can be implemented in devices with limited storage and processing capabilities.

Multi-agent architectures and frameworks such as OAA [4], RETSINA [5] and JADE [6] define agent-based structures to resolve distributed computational problems and facilitate user interactions. However, the communication capabilities are limited, not allowing easy integration with services and applications. On the other hand, SOA-based architectures usually do not provide intelligent computational and interactive mechanisms. FUSION@ combines both paradigms, trying to take advantage of their strengths and avoid their weaknesses. FUSION@ sets on top of existing agent frameworks by adding new layers to integrate a service-oriented approach and facilitate the distribution and management of resources. Therefore, the FUSION@ framework has been modeled following the SOA model, but adding the applications block which represents the interaction with users. FUSION@ defines four basic blocks: *Applications*, *Agent Platform*, *Services* and *Communication Protocol*. These blocks provide all the functionalities of the architecture [15].

One of the advantages of FUSION@ is that the users can access the system through distributed applications, which run on different types of devices and interfaces (e.g. computers, cell phones, PDAs). All requests and responses are handled by the agents in the platform. The agents analyze all requests and invoke the specified services either locally or remotely. Services process the requests and execute the specified tasks. Then, services send back a response with the result of the specific task. Moreover, the platform makes use of deliberative BDI (Belief, Desire, Intention) agents to select the optimal option to perform a task, so users do not need to find and specify the service to be invoked by the application. These features have been introduced in FUSION@ to create a secure communication between applications and services. There are different kinds of agents in the architecture, each one with specific roles, capabilities and characteristics. Thus, there are pre-defined agents which provide the basic functionalities of the architecture. These pre-defined agents come from the sequential stages followed using the Gaia methodology and the SysML language.

On the Gaia analysis stage it is defined the Roles Model and the Interaction Model from the characteristics and requirements of the system. This way, it is

defined the roles that collaborate in the system, as well as the protocols and activities that must carry out each of the roles. The Gaia high-level stage includes the Agent Model, the Services Model and the Acquaintance Model. The Agent Model allows determining what agents should implement the roles defined before. The Acquaintance Model lets the designer realize the relationships existing among the agents to be implemented. Fig. 1 shows two Gaia analysis and design models of FUSION@. Fig. 1a depicts the Role Model for the Role Administrator, while Fig. 1b shows the Interaction Model of the processing of a service request on which three roles are involved: Application Communicator, Administrator and Service Communicator. From the Acquaintance Model we obtain the pre-defined agents: *Admin*, *Directory*, *CommApp*, *CommServ*, *Interface*, *Security* and *Supervisor* [15].

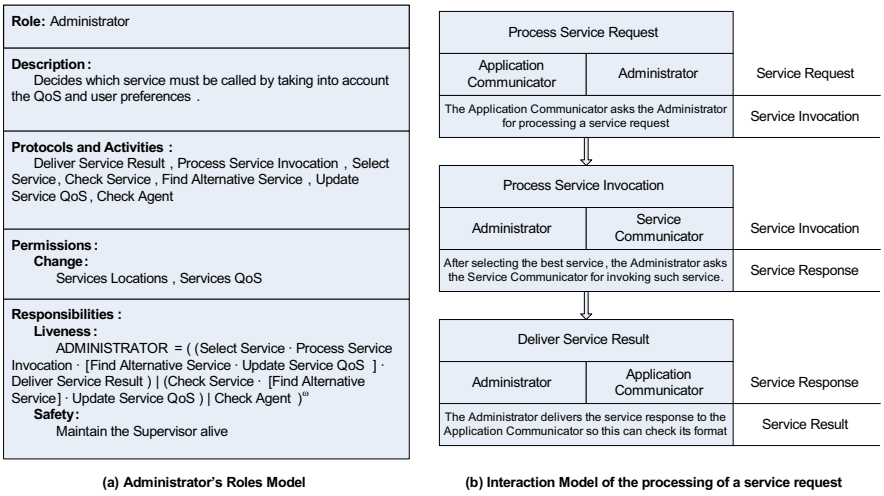

Fig. 1 Administrator's Roles Model and the Interaction Model for a service request processing

After the analysis and high-level design phases of the Gaia methodology, designers may go on analyzing the results and carry out a low-level design by means of SysML. This stage provides the Blocks Definition Diagrams, the Sequence Diagrams, the Interaction Diagrams and the State Diagrams. Each Blocks Definition Diagram represents an agent on the system, describing its capacities and services that must implement, as can be seen in Fig. 2 for the Admin Agent. Sequence Diagrams and Interaction Diagrams represent interactions among agents, the first over the time and the latter emphasizing the associations existing among them. Fig. 3 shows the State Diagram of the Admin Agent. Thus, Fig. 3 describes the possible states on which the Admin Agent can be and what events make the agent to jump from one state to another. As can be seen, this information is very useful for the developers who have to finally encode the agents.

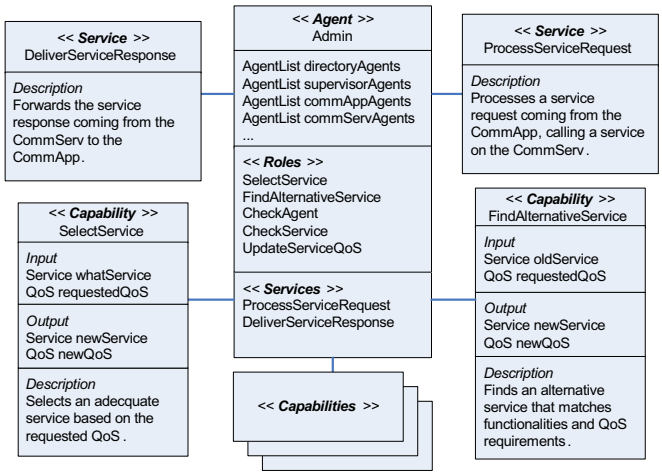

Fig. 2 Admin Agent's SysML Block Definition Diagram

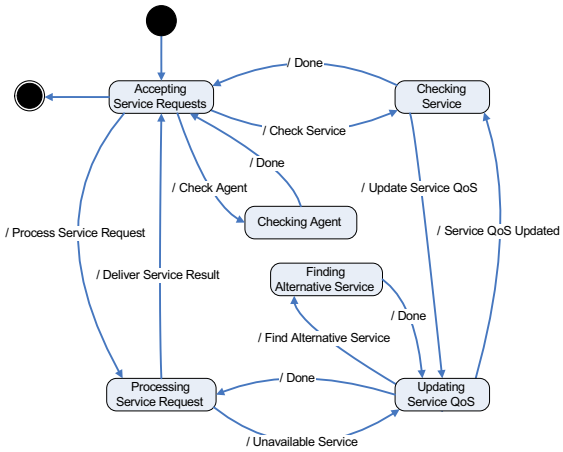

Fig. 3 Admin Agent's State Diagram

After analyzing and designing the architecture, it is possible to describe its components by means of a certain ontological language. To describe the components of FUSION@, the OWL language has been chosen, making use of the RDF/XML rendering. This choice has been motivated by the fact that OWL is especially designed for being used on the Web and can utilize the RDF and XML standards. This will be very useful for an architecture that joins Web services and intelligent agents, so it is intended to manage knowledge about itself in the future. This is a fragment of the resulting ontology[1]:

[1] For the whole ontology of FUSION@, please contact with BISITE at http://bisite.usal.es

```
<owl:Class rdf:about="#adminAgent">
   <owl:equivalentClass>
      <owl:Class>
         <owl:intersectionOf
rdf:parseType="Collection">
            <rdf:Description rdf:about="#agent"/>
            <rdf:Description
rdf:about="#administrator"/>
         </owl:intersectionOf>
      </owl:Class>
   </owl:equivalentClass>
</owl:Class>
```

FUSION@ is an open architecture that allows developers to modify the structure of the agents described before. Developers can add new agent types or extend the existing ones to conform to their projects needs. However, most of the agents' functionalities should be modeled as services, releasing them from tasks that could be performed by services. All information related to services is stored into a dynamic directory which the platform uses to invoke them. Services are requested by users through applications, but all requests are managed by the platform. This provides more control and security when requesting a service because the agents can control and validate all messages sent to the services and applications. Developers are free to use any programming language, although they must follow the communication protocol based on transactions of XML (SOAP) messages.

4 Results and Conclusions

As a conclusion we can say that although FUSION@ is still under development, preliminary results demonstrate that it is adequate for building complex systems and exploiting composite services [15]. FUSION@ has laid the groundwork to boost and optimize the development of future projects and systems that combine the flexibility of a SOA approach with the intelligence provided by agents. The distributed approach of FUSION@ optimizes usability and performance because it can be obtained lighter agents by modeling the systems' functionalities as independent services and applications outside of the agents' structure, thus these may be used in other developments.

Thanks to the employment of AOSE tools such Gaia and SysML, the analysis and design stages of the development of FUSION@ have made it possible to achieve a model close to its implementation. This have implied a shorter development time, allowing designers to focus on obtaining a well-designed distributed and dynamic architecture, rather than on encoding and debugging tasks. Furthermore, the ontology of FUSION@ will allow managing knowledge about the entities involved on it and make it possible to exchange information about agents themselves through local and remote services linked to the architecture. Our experience in the design and development of multi-agent systems has demonstrated us that the use of AOSE tools not only reduces the development time, but also improves the quality of the final system, as well as its scalability and reliability.

Acknowledgments. This work has been supported by the Ministerio de Ciencia e Innovación project PET2008_0036.

References

[1] Cerami, E.: Web Services Essentials: Distributed Applications with XML-RPC, SOAP, UDDI & WSDL, 1st edn. O'Reilly Media, Inc., Sebastopol (2002)
[2] Wooldridge, M.: An Introduction to MultiAgent Systems, 2nd edn. Wiley, Chichester (2009)
[3] Ardissono, L., Petrone, G., Segnan, M.: A conversational approach to the interaction with Web Services. Computational intelligence 20(4), 693–709 (2004)
[4] Martin, D.L., Cheyer, A.J., Moran, D.B., et al.: The open agent architecture: A framework for building distributed software systems. Applied Artificial Intelligence 13(1), 91–128 (1999)
[5] Sycara, K., Paolucci, M., Van Velsen, M., Giampapa, J.: The RETSINA MAS Infrastructure. Autonomous Agents and Multi-Agent Systems 7(1), 29–48 (2003)
[6] Bellifemine, F., Poggi, A., Rimassa, G.: JADE–A FIPA-compliant agent framework. In: Proceedings of PAAM, vol. 99, pp. 97–108 (1999)
[7] Rigole, P., Holvoet, T., Berbers, Y.: Using Jini to integrate home automation in a distributed software-system. In: Plaice, J., Kropf, P.G., Schulthess, P., Slonim, J. (eds.) DCW 2002. LNCS, vol. 2468, pp. 291–303. Springer, Heidelberg (2002)
[8] Wooldridge, M., Jennings, N.R., Kinny, D.: The Gaia methodology for agent-oriented analysis and design. Autonomous Agents and Multi-Agent Systems 3(3), 285–312 (2000)
[9] SysML - Open Source Specification Project (Recovered, October 2009), http://www.sysml.org/
[10] Pan, J.Z., Horrocks, I.: RDFS(FA): Connecting RDF(S) and OWL DL. IEEE Transactions on Knowledge and Data Engineering 19(2), 192–206 (2007)
[11] Bonino da Silva, L.O., Ramparany, F., Dockhorn, P., Vink, P., Etter, R., Broens, T.: A service architecture for context awareness and reaction provisioning. In: IEEE Congress on Services (Services 2007), pp. 25–32 (2007)
[12] Ricci, A., Buda, C., Zaghini, N.: An agent-oriented programming model for SOA & web services. In: 5th IEEE International Conference on Industrial Informatics, pp. 1059–1064 (2007)
[13] Gruber, T.R.: A translation approach to portable ontology specifications. Knowl. Acquis. 5(2), 199–220 (1993)
[14] Hendler, J.: The Dark Side of the Semantic Web. IEEE Intelligent Systems 22(1), 2–4 (2007)
[15] Tapia, D., Bajo, J., Corchado, J.: Distributing Functionalities in a SOA-Based Multiagent Architecture. In: 7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009), pp. 20–29 (2009)