Web-MASI : Multi-Agent Systems Interoperability Using a Web Services Based Approach

Tarak Melliti
Serge Haddad
LAMSADE, University of Paris Dauphine
{melliti, haddad}@lamsade.dauphine.fr

Alexandru Suna
Amal El Fallah Seghrouchni
LIP6, University of Paris 6
{Alexandru.Suna,Amal.Elfallah}@lip6.fr

Abstract

In this paper we present a conceptual and architectural framework for the multi-agent systems' interoperability based on Web services. Agents publish their abilities as Web Services that can be used by other agents, independently of conceptual (e.g. architecture) or technical (e.g. platform, programming language) aspects. The proposed architecture and concepts have been tested and validated using the CLAIM language and the SyMPA platform.

1. Introduction

In this paper we present a conceptual and architectural framework for the interoperability of heterogenous multi-agent systems (MAS). The main objective of the interoperability is to allow communication and cooperation between several software systems (similar or heterogenous) in order to reduce the integration cost. We distinguish three levels of interoperability: the technological level, the syntactical level and the semantical level.

As software systems, MAS are equally confronted with the interoperability issue, more especially as their vocation to autonomously interact and cooperate is the essence of their existence. The cognitive agents often exhibit a goal-driven behavior. So interoperability cannot be limited to a simple service invocation but requires the development of: 1) sophisticated mechanisms for research of services able to satisfy current needs; 2) complex interaction models.

The MAS interoperability has been promoted by various institutions such as FIPA,[1] which deploys enormous efforts for proposing standards ensuring interoperability at various levels of the MAS architectures [3]. We can mention FIPA-ACL at the communication level, DAML+OIL [9] at the semantical level, AAA[2] at the architectural level, etc.

1 Foundation for Intelligent Physical Agents: http://www.fipa.org/
2 Abstract Agent Architecture

Nevertheless, instead of ensuring interoperability between heterogenous MAS, the interoperability is conditioned by the compatibility of the MAS with the FIPA specifications. Moreover, the level of cognition required by the FIPA specifications (*e.g.* FIPA-ACL) makes difficult the interoperability between heterogeneous MAS.

In the software engineering domain, as testified by the normalization and development efforts concerning the Web Services [1], they offer today a credible support allowing applications to expose their functionalities through standardized and increasingly experimented interfaces.

We witness today a connection between the MAS and the Web Services, following several directions:

1. Using MAS as a mediating entity in the functional model of the Web Services [7].
2. Using Web Services as a technological and architectural framework for building MAS accessible through the Web. In this type of application, agents offer their capabilities through Web Services. We distinguish two categories:

- An integrated conception: the Web Services are developed following an agent model, in order to carry out complex tasks (*e.g.* [5]).
- An uncoupled conception: starting from a MAS given *a priori*, a Web Services based layer makes agents' capabilities accessible through the Web for other agents or for traditional client applications [6].

In this paper we propose an uncoupled conception. Comparing with [6], which is limited to a syntactic translation from an interface language to another one (SD towards WSDL), our approach includes the following original aspects:

- Algorithms for the synthesis of the service's behavior and of a client supporting the functionalities of the interaction protocol.
- A method allowing to identify the parts to be rewritten and to choose the level of integration.

Based on Web Services, the interoperability environment we propose, named Web-MASI (Web-MAS Interoperability) relies on two elements: an architecture which encapsu-

lates the MAS in the functional model of the Web Services and an interoperability module which constitutes the interface between the MAS and the Web Services environment. The proposed architecture as well as the introduced concepts were tested and validated using the CLAIM language and the SyMPA platform [2].

2. Web Services at a glance

2.1. Functional model

The Web Services model relies on a service oriented architecture [1] based on three main categories of actors: the *service providers* (*i.e.* entities responsible for the Web Service), the *clients* acting as intermediaries for the services' users and the *directories* offering to suppliers the possibility to publish their services and to customers the means for locating their requirements in term of services.

The architecture's dynamics can be decomposed in: publication of the service's description, the localization of the service and the service's invocation. This dynamics is normalized through several standards: **SOAP** - an abstract protocol for describing and structuring messages, **UDDI** - an XML specification for directories and **WSDL** - a format for the description of the Web Services published in directories. A WSDL service is composed of a set of elementary operations, each one described by the message flow exchanged between the client and the service.

2.2. Complex Web Services: operational and semantic aspects

However, the current state of the Web Services model presents conceptual limits that lead to two main complementary research directions. On the one hand, the directories taxonomies are not expressive enough to allow a fine correspondence between the user's needs and the available services. In this direction, the current researches belong to the semantic Web domain (*e.g.* OWL-S). On the other hand, the WSDL semantics is not expressive enough to support certain types of services requiring a long interaction, controlled by an explicit model of process. Several extensions of WSDL were proposed in order to support complex Web Services, such as Xlang, WSFL, BPEL4WS, etc. [8]. These languages propose a set of operators which apply in a modular way to the basic units of the exchange of messages. In this work we are using Xlang for describing the service's behavior. Here are the main Xlang constructors. The notations $!o[m]$ and $?o[m]$ represent the WSDL operations for message transmission and reception, while $r[e]$ represents the raising of the exception e.

- $P;Q$ sequentially executes P followed by Q.

- $switch[\{P_i\}_{i\in I}]$: following an internal choice, the service behaves in a non deterministic manner executing one of the P_i processes.
- $pick[\{(m_i, P_i)\}_{i\in I}, d, Q]$: if the service receives a message m_i, it executes P_i. If none of the m_i messages is received in d units of time, it behaves as Q.
- $context[P, E]$ constructor defines a service P guarded by certain events defined in the set of events E.

The observable description of the service makes (by construction) its behavior non deterministic, which makes the interaction process non trivial. In a previous paper [4], we tried to solve this problem. First, we defined an operational semantics for Xlang in order to represent the service's behavior as a timed automata. Then, we defined an interaction relation that must verify the client and the Web Service. Finally, we designed an algorithm which, given the service's behavior description (Xlang or others), either builds a correct client or detects the service's ambiguity. This work also provided an implementation of a generic client able to interact with complex Web Services.

3. Integrating heterogenous MAS

3.1. Interoperability architecture

The interoperability environment we propose (figure 1), Web-MASI, relies on two key elements: an architecture which encapsulates the MAS in the functional model of the Web Services and an interoperability module which constitutes the interface between the MAS and the Web Services environment. In our interoperability architecture, agents are in the same time providers and consumers of services. They use UDDI directories for publishing their capabilities in order to be discovered and used by other agents, in a modular and uniform way. The interoperability module (IM) offers, to various MAS, tools for synthesizing, publishing, locating and invoking Web Services. The service's invocation and execution are carried out by our generic client contained in the IM.

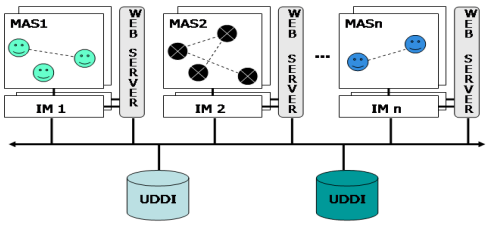

Figure 1. Environment's architecture

3.2. Requirements

We made efforts for minimizing the functional requirements on the MAS in order to minimize the integration cost and consequently to cover a wide range of MAS models. Nevertheless, these MAS must present certain characteristics. At the MAS level, every agent must have a unique identity. This requirement is essential for the correct behavior of the IM. At the agent level, the requirements relate mainly to the conditions of the interaction initiative and process. The elements which play a part in the agent's execution are the *goals*, the *capabilities* (tasks provided with a description covering the aspects related to its effects, invocation and execution (see 3.3.1)) and the initiative for proposing and requesting capabilities. The functional requirements are defined at an abstract level.

3.3. The dynamics of the interoperability environment

The interoperability module (IM) we propose must be deployed on every computer hosting agents. It is composed of two submodules which respectively achieve the function of publication (P) and the functions of research and invocation (RI) of service. Figure 2 presents the IM dynamics.

a) Publication (P)
1) An agent A_i decides to publish one (or several) of his capabilities (C_1 on the figure).
2) Starting from the capability's description, the module P generates a Web Service (A_iC_1) and deploys it on the local Web server (see 3.3.1).
3) The service's description is published using UDDI.

b,c) Research and invocation (RI)
4) An agent (A_j) uses the RI module's functionalities in order to find services satisfying his needs.
5) The RI module questions the UDDI directories.
6) The RI module receives from the directories a set of Web Services corresponding to its search criteria.
7) The RI module autonomously starts the invocation of the discovered services in order to find their various attributes.
8) The RI module sends the located services to the agent.
9) The agent (A_j) selects a service.
10) The RI module concurrently sends confirmation messages to the chosen services and annulation messages to the other considered services.
11) The service receiving the activation message, translates it in a format understandable by the agent.
After the successful execution of the capability, the requesting agent is informed of the result using the same IM.

One of the strong features of our architecture is that the deployed Web Service is not limited to the capability's invocation, but integrates the negotiation phase and the execution tracking. This is made possible by the algorithm of service synthesis.

3.3.1. Informal description of the algorithm for service synthesis.

The synthesis algorithm takes as input an XML description of the capability and of its attributes. We can distinguish three categories of elements composing this description.

Agent's description containing the identity of the agent providing the capability, as well as optional additional information.
Capability's description containing a semantic description composed of four XML elements, such as its name, its pre and post conditions and the messages necessary for the interaction process. The postcondition is the key element for the research in the UDDI directories.
Interaction's description is optional but improves the conditions of the interaction. It is composed of static and dynamic attributes. The static attributes define the aspects of the capability execution known *a priori*, such as temporal constraint, the possibility of annulation, etc. The values of the dynamic attributes (*e.g.* agent's availability, QoS, etc.) are established at the interaction time.

The invocation of the corresponding capability has two phases: a negotiation phase and capability's execution phase.

1. The negotiation phase starts when the service receives a message requesting an interaction. The aim of this phase is to obtain the dynamic attributes' values. Here is a generic example concerning this phase :

$$negotiation : ?o[request\_interaction];$$
$$switch(!o[attributes]; execution, !o[reject])$$

2. The execution phase. The Xlang service depends on the capability's description. The interaction necessary for the execution of the capability is encapsulated in a $context$ block controlled by events generated from attributes such as cancelling messages, timeouts, etc. The body of the $context$ block consists in the flow of messages of the capability's execution, *e.g.*:

$$execution : context[?o[invocation];$$
$$switch(!o[result]; !o[problem]),$$
$$[pick[(cancel, !o[abort]), (delay\_max, !o[timeout])]$$

For every capability description the synthesis algorithm generates the two phases Web service and the corresponding WSDL-Xlang files (published in UDDI).

4. Experiments using CLAIM and SyMPA

In this section we present the integration process performed in order to validate our approach. CLAIM [2] is an agent oriented programming language allowing to design

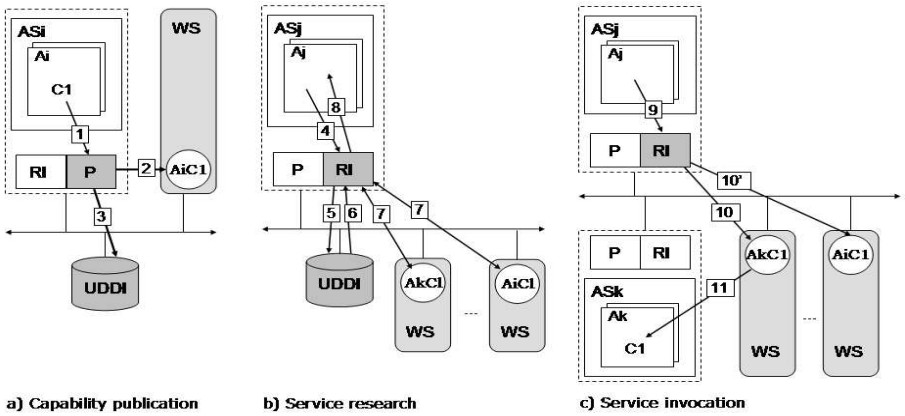

Figure 2. Interoperability Module's dynamics

and implement intelligent and mobile agents distributed on several connected computers. CLAIM agents meet the requirements defined in section 3.2. They are guided by goals, that can be achieved by executing capabilities. The language is supported by a distributed platform, called SyMPA [2]. The integration process consisted in developing an additional layer to SyMPA, an interface toward the interoperability module (IM).

Concerning the *publication* phase, we developed a method generating a Web-MASI description of a CLAIM capability.

For *researching* services, without the IM, CLAIM offers to agents mechanisms for presenting their capabilities or for searching and requesting capabilities from other CLAIM agents. Adding the IM, an agent can publish his capabilities and can requests services from other agents, belonging to other platforms or MAS, deployed as Web services.

When an agent choose a Web Service, he uses the IM's functionalities to *invoke* the service.

The *execution* of a published capability is directed influenced by the elements created during the publication phase. When a deployed Web Service receives an invocation SOAP message, it has to forward the invocation in a CLAIM format to the concerned agent.

In conclusion, adapting the IM for CLAIM and SyMPA proved to be easy. This integration required the development of small conversion modules between the Web-MASI format and the CLAIM format and the integration in the agents' code of the invocations of the provided APIs.

5. Conclusion and future work

This paper presented a conceptual and architectural framework based on Web Services for the interoperability of heterogenous MAS. The proposed interoperability environment, Web-MASI, is composed of two elements: an encapsulation of MAS in the functional model of the Web Services and an interoperability module which constitutes the interface between the MAS and the Web Services environment. Concerning the future work, as a short time objective, we intend to integrate OWL-S ontologies in the semantic description of the agents' goals.

References

[1] F. Curbera, W. A. Nagy, and S. Weerawarana. Web services: Why and how? *OOPSLA 2001 Workshop on Object-Oriented Web Services*, 2001.

[2] A. El Fallah Seghrouchni and A. Suna. Claim: A computational language for autonomous, intelligent and mobile agents. *LNAI*, 3067:90–110, 2004.

[3] B. R. A. Flores-Mendez. Standardization of multi-agent system frameworks. *ACM Crossroads, 5(4)*, 1999.

[4] S. Haddad, T. Melliti, P. Moreaux, and S. Rampacek. Modelling web services interoperability. In *Proceedings of the Sixth International Conference on Entreprise Information Systems*, pages 287–295, Porto, Portugal, 2004.

[5] T. Jin and S. Goschnick. Utilizing web services in an agent based transaction model (ABT). In *Workshop on Web services And Agent-based ingineering*, Melbourne, Australia, 2003.

[6] M. Lyell, L. Rosen, M. Casagni-Simkins, and D. Norris. On software agents and web services: Usage and design concepts and issues. In *Workshop on Web services And Agent-based ingineering*, Melbourne, Australia, 2003.

[7] D. Richards, S. van Splunter, F. M. Brazier, and M. Sabou. Composing web services using an agent factory. In *Workshop on Web services And Agent-based ingineering*, Melbourne Australia, 2003.

[8] S. Staab, W. van der Aalst, V. R. Benjamins, A. Sheth, J. A. Miller, C. Bussler, A. Maedche, D. Fensel, and D. Gannon. Web services: Been there, done that? *IEEE Intelligent Systems: volume 18*, pages 72–85, 2003.

[9] The OWL Services Coalition. OWL-S, Mars 2003. http://www.daml.org/services.