Improving Web services adaptability thanks to a synergy between aspect programming and a multi-agent middleware

Flavien BALBO¹, Valérie MONFORT^{1,2}

¹ University Paris-Dauphine - LAMSADE,
Place du Maréchal de Lattre de Tassigny,
F-75775 Paris 16 Cedex, France.
balbo@lamsade.dauphine.fr

² Paris 1 -Panthéon- Sorbonne University,
Centre de Recherche en Informatique
90 rue de Tolbiac 75634 Paris cedex 13, France.
valerie.monfort@univ-paris1.fr

Abstract—Many companies aim to use Web services to integrate heterogeneous and/or remote applications in SOA (Service Oriented Architecture) contexts. We noticed the code of the Web service is monolithic, mixing concerns, and difficult to maintain. On top of all, any change requires to re deploy the modified Web service. Business rules are very often spread out in applications code. Faced to these problems, we propose: 1) To improve the adaptability of Web services thanks to the aspect based service paradigm; 2) To improve adaptability for services coordination thanks to an Multi-Agent System. To explain our architecture, we use a concrete and pragmatic example.

Keywords-Aspect Based services, Multi Agent System, SOA, Web services, BPEL.

I. INTRODUCTION

Economical context impacts companies and their Information Systems (IS). Companies acquire other competitors or develop new business skills, delocalize whole or part of their organization. Their IS are faced to these complex evolutions and have to overcome these changes. In this context, Service Oriented Architecture (SOA) offers a great flexibility to IS. Applications are seen as black boxes independently connected to an application as Enterprise Application Integration bus (EAI) with its connectors. However, this integration solution does not allow connecting heterogeneous applications or infrastructures. Web services (WS) are based on standards and they are the cheaper and simplest solution to resolve this problem. Enterprise Services Bus (ESB) allows increasing loosely coupling with low costs. However, we noticed several limitations concerning WS and their coordination by an ESB like the lack of adaptability.

In this paper, we propose a complete solution to improve the flexibility in the use of WS. Firstly, we propose to improve the adaptability of a web service by aspect based service paradigm [1]. The aim is the design of WS that are adaptable without intrusive modification of their implementation. Secondly, we propose a Multi-Agent System (MAS) to improve the dynamic of the coordination process of WS [2]. The aim is to improve the loosening coupling coordination mechanism thanks to a middleware that is itself dynamically adaptable.

The section II presents the fundamentals of the relations between WS and multi-agent coordination according to the adaptability point of view. Thanks to an example, section III proposes our solution where the environment of the MAS is the cornerstone of the communication between agents. Then, section IV presents our conclusion and future works.

II. MULTI-AGENT, WS AND ADAPTABILITY

A. Web-services and adaptability

Web services (WS), like any other middleware technologies, aim to provide mechanisms to bridge heterogeneous platforms, allowing data to flow across various programs. The winning card of this technology is not its mechanism but rather the standards upon which it is built (WSDL, SOAP, UDDI). For this reason, WS are highly interoperable and do not rely on the underlying platform they are built on. WS provide a minimalist mechanism to interconnect different applications. But one fundamental point is the importance of the WSDL being the exact interface of the system. With WS, the message contract (WSDL) is the central meeting point which connects applications. The WSDL contract constitutes the design view upon which developers can generate both client and server sides (proxy and stub). Even if WS are the fittest current solution for interoperability in companies IS and SOA implementation, we noticed several limitations as a very monolithic code encapsulating different concerns as business, security,... Moreover, we used to change WS code according to new needs, to redeploy WS. Each change is time costly and WS is not available. It is the reason why we used Aspect Oriented Programming (AOP) to increase flexibility of SOA.

WS-BPEL is a WS orchestration language. An orchestration specifies an executable process that involves message exchanges with other systems, such that the message exchange sequences are controlled by the orchestration designer. WS-BPEL provides a language for the specification of executable and abstract business processes. By doing so, it extends the WS interaction model and enables it to support business transactions. WS-BPEL defines an interoperable integration model that facilitates the expansion of

automated process integration in both the intra-corporate and the business-to-business spaces.

The issue is that BPEL is an XML-based language that is not adapted to dynamic modification of the coordination process. More often business rules are imbedded into the process or directly into the application code. Any modification becomes complex or simply impossible according to timing. To overcome these limits, we propose to coordinate WS thanks to an agent approach that provides more or less middleware functionalities and knowledge capitalization and mastering.

B. Multi-agent, Web services and coordination

Numerous researches have been done to combine both Multi-Agent Systems (MAS) and Web services: the WS technology proposes a standardized distribution of functionalities and agent technology proposes higher-lever software entities with cognitive abilities to better use these functionalities. Following the coordination viewpoint, the relation between WS and MAS belongs to one of these categories: 1) Interfaces are designed to support the interoperability between the two technologies and the coordination depends of the agent and/or WS design [3], [4], [5]; 2) Interface agents are designed to invoke the WS and belonged to an MAS that is designed to coordinate WS [6], [7].

Our proposition belongs to the second category where an MAS becomes a middle component in the WS functional model [6], [7]. In this category, an Interface Agent (IA) is associated to each WS and the coordination between the WS depends of the coordination between the IA. There are two main approaches in the design of the solutions according to the location of the execution and control of the process related to the coordination of WS. In the first approach, an agent executes and controls the process; the MAS is designed to simplify the agent execution thanks to adapted services [7]. In the second approach, the MAS executes, controls the process; the MAS is designed to support the coordination process [6]. These current solutions are not adapted to dynamic modifications of the initial process because each modification implies a modification of the agents. From our point of view, one solution to overcome this limit is to separate the agent design from the coordination process design.

III. PROPOSITION

Our proposition has two complementary components. Firstly, we propose to design Web-services with Aspect to increase flexibility and adaptability, secondly we propose a middleware based on the multi-agent paradigm that is designed to support a dynamic adaptability process of the WS. Business rules management is mastered by the second solution because rules are intrinsic to agent model.

A. Applying AOP to WS: Aspect Service Weaver (ASW)

Many researches consider AOP as an answer to improve WS flexibility. AOP is a paradigm that enables the modularization of crosscutting concerns into single units called aspects, which are modular units of crosscutting implementation. Aspect-Oriented languages are implemented over a set of definitions: 1) *Joinpoints*: They denote the locations in the program that are affected by a particular crosscutting concern. 2) *Pointcuts*: They specify a collection of conditional joinpoints. 3) *Advices*: They are codes that are executed *before*, *after* or *around* a joinpoint. In AOP, a tool named *weaver* takes the code specified in a traditional (base) programming language, and the additional code specified in an aspect language, and merges the two together in order to generate the final behavior. The weaving can occur at compile time (modifying the compiler), load time (modifying the class loader) or runtime (modifying the interpreter).

In our previous approach, we developed an AOP based tool named *Aspect Service Weaver (ASW)* [1]. The ASW intercepts the SOAP messages between a client and an elementary WS, then verifies during the interaction if there is a new behavior introduced (*advice services*). We use the AOP weaving time to add the new behavior. So we proposed a solution based on AOP to handle dynamic changes in the WS (client side and server side both) with a non intrusive manner and by respecting standards.

B. Functional middleware description

Most of the multi-agent coordination models are based on interaction protocols that rely on the exchange of messages between agents. The objective of our middleware is to improve the control by the agents of the message exchanges. The result is the design of new interaction models: our middleware supports multi-party communication (MPC) and contextual interaction. The advantage of MPC is to overcome the limits of the dyadic interaction. The advantage of contextual interaction is to adapt the interaction process to the state of the MAS.

Our middleware that belongs to the field of distributed agent environment is based on the EASI (Environment as Active Support of Interaction) model [2]. The EASI model enables agents to use the environment to exchange messages and, more precisely, it enables an agent to send messages to another agent that is located by the environment and it enables agents to perceive every exchanged message.

Following the EASI model, the environment records in the Description Module (Fig. 1) the Symbolic Descriptions (SD) of the agents and messages. A SD is a set of property-value pairs that describes the agent or the message. Each agent description is updated by the agent itself. To find the right receiver of a message, the environment contains Symbolic Filters (SF) that are stored and triggered in the Filter sub-module. A Symbolic Filter is a set of constraints on the properties of the SD that defines the context of an

Figure 1. The Environment functional description

elementary interaction. There are two types of SF, the first enables the perception of a message (the message remains in the environment) and the second enables its reception (the message is deleted from the environment). Except contrary mention, the SF enable the perception of messages in the following. The SF are put dynamically by the agents in the environment. A message put in the environment is stored in the Message Box sub-module; a SD is created and stored in the Description Module. Thanks to its Symbolic Description, the message will be perceived or received by every agent that has a Symbolic Filter that is matched in the current informational context. The filter triggering is the beginning of a contextual elementary interaction process that ends with the processing of the message by the receiver agent. The combination of these contextual elementary interaction processes enables the coordination of the agents following the data-oriented coordination models [8].

The dynamic management of their SD enables to the agents to adapt the interaction process to their state. For example, if the SD of an agent contains information about its availability then this agent can in its SF limit the perception of messages only when it is available. Processed by the environment, the interaction can be dynamically changed with the addition/retraction of the SF in the environment without changing the implementation of the agent that processes the interaction. Contrary to the other discussed solutions, the coordination process is supported by the environment thanks to SF and SD, and not by the agents. By comparison, the agents like WS give functionalities and the environment like ESB supports the coordination. The difference is that agents can change dynamically the coordination process by changing their SD or their SF.

C. Example

The following example has been proposed in [6]: This service provides the ability to obtain a stock quote from

the NYSE in any currency. This service is composed from three other WS: the stockQuoteProvider (SQP), the currencyExchangeProvider (CEP) and the simpleFloatMulti-Provider (SFMP). A narrative description of the steps in the workflow follows: 1. A service request message is received from the requestor; 2. The stock symbol is copied from the request message and SQP is invoked to obtain the quote; 3. The country name is copied from the request message and CEP is invoked to obtain the rate of exchange between US dollars and the foreign currency; 4. The returned quote and the exchange rate are sent to SFMP, which multiplies these values to convert the quote to the foreign currency; 5. The converted quote is copied into the reply message which is sent to the service requester.

In order to coordinate the web services, interface agents (IA) that control the WS invocation are designed: the Client Agent (CA) which processes the request of the human user; the Stock Agent (SA) for SQP; the Exchange Agent (EA) for CEP and the Multiplication Agent (MA) for SFMP.

At the beginning of the process (Fig. 2), the environment contains the SD of the interface agents and their SF to execute the service:

- 1) Client Agent (CA), $\{sd_{CA}, sf_{CA}\}$: sd_{CA} contains the identifier (*id*) of the CA. This agent adds its *id* value to the *request* message of the human user that contains information related to the stock symbol and the country name. It put the result in the environment (0). The symbolic filter sf_{CA} enables the reception of messages according to this information (6).
- 2) Stock Agent (SA), $\{sd_{SA}, sf_{SA}\}$: sd_{SA} contains the *capability* of SA and sf_{SA} enables to receive the *request* messages related to a stock (1). After the processing of the answer, SA put in the environment an *inform* message (2) that is addressed to CA.
- 3) Exchange Agent (EA), $\{sd_{EA}, sf_{EA}\}$: sd_{EA} contains the *capability* of EA and sf_{EA} enables to receive the *request* messages (1) related to an exchange rate. After the processing of the answer, EA put in the environment an *inform* message (2) that is addressed to the CA.
- 4) Multiplication Agent (MA), $\{sd_{MA}, sf_{MA}\}$: sd_{MA} contains the *capability* of MA and sf_{MA} enables to receive the *request* messages (1) related to a stock and a country name. For a received *request* message, it put in the environment a new symbolic filter called sf_{MA-Id} that enables to receive the *inform* messages related to a stock and an exchange rate and addressed to the sender of the received *request* message. When the *inform* messages are put in the environment by the Stock Agent and the Exchange Agent, sf_{MA-Id} is triggered and the Multiplication Agent receives the messages (4). It put in the environment an *inform* message that is addressed to the Client Agent.

Figure 2. Interaction between agents mediated by the environment

A modification in this process with the addition of new services with their associated IA will be done with the addition of the associated SD and SF. For example, a tax service that modifies the rate of exchange according to the foreign currency is added; TA is the interface agent of this WS. The addition of the Tax Agent (TA) changes the result of the initial process. The issue is that this agent has to complete the result given by EA before MA. To do it, EA puts in the environment its SD (sd_{EA}) that contains its *capability* and a SF (sf_{EA}) to receive the *inform* messages put by the Exchange Agent in the environment (3). When it receives an *inform* message, it adds to the value related to the change rate the tax related to the country and put an *inform* message (4) in the environment with this new value.

This coordination process depends of the order of the filters' triggering. Following the EASI model, the SF are triggered according to a priority level that is given by the designer. If the symbolic filters of the initial process are ordered in a decreasing order then the symbolic filter sf_{MA-ID} has a priority value lower than sf_{EA} . If sf_{TA} has a value between the values of these two symbolic filters the issue is solved. TA receives the *inform* message that is deleted from the environment to be replaced by the *inform* message of the TA.

To be complete, our proposition has to take into account the WS that are invoked by IA and that are themselves composed WS. If a single service or a complex service do not fit anymore to the needs Aspect based services are used to re route message sending as shown previously. According to previous example, a new requirement asks to any stocks management to use identification aspects before any transaction and security aspect after any transaction only for some users. But other users go on using previous services

without any impact and any redeployment. Aspects come from IS part and are required by Interface Agent to IS. So, aspect based services are used for rerouting.

IV. CONCLUSION

SOA aims to build components based architecture offering services. It is the solution allowing best flexibility by considering applications as black boxes accessible by their services based interfaces. WS, as built on standards, are the current most fitted solution to connect applications and to get interoperability, in and between, heterogeneous IS. Nevertheless, this solution is not so flexible as expected. Moreover business rules are spread out all over applications. We propose service based MAS as an adaptable middleware and rules management solution. We suggest to allow a connection to the rest of IS with Interface agents. They communicate with SOA based IS, where services are simple or complex (orchestrated). We also propose to increase adaptability by adding aspect based services by re routing messages, till now, outside environment. We are convinced that our future works have to increase MAS adaptability by adding aspects in the MAS infrastructure.

REFERENCES

- [1] R. F. Tomaz, M. B. Hmida, and V. Monfort, "Concrete solutions for web services adaptability using policies and aspects," *Int. J. Cooperative Inf. Syst.*, vol. 15, no. 3, pp. 415–438, 2006.
- [2] J. Saunier and F. Balbo, "Regulated multi-party communications and context awareness through the environment," *Multi-agent Grid Syst.*, vol. 5, no. 1, pp. 75–91, 2009.
- [3] D. Greenwood, M. Lyell, A. Mallya, and H. Suguri, "The ieee fipa approach to integrating software agents and web services," in *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. New York, NY, USA: ACM, 2007, pp. 1–7.
- [4] T. Melliti, S. Haddad, A. Suna, and A. E. F. Seghrouchni, "Web-masi: Multi-agent systems interoperability using a web services based approach," in *IAT '05: Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 739–742.
- [5] A. S. Namin, W. Shen, and H. Ghenniwa, "Implementing enterprise collaboration using web services and software agents," *Int. J. Netw. Virtual Organ.*, vol. 3, no. 2, pp. 185–201, 2006.
- [6] P. A. Buhler and J. M. Vidal, "Towards adaptive workflow enactment using multiagent systems," *Inf. Technol. and Management*, vol. 6, no. 1, pp. 61–87, 2005.
- [7] F. García-Sánchez, R. Valencia-García, R. Martínez-Béjar, and J. T. Fernández-Breis, "An ontology, intelligent agent-based framework for the provision of semantic web services," *Expert Syst. Appl.*, vol. 36, no. 2, pp. 3167–3187, 2009.
- [8] D. Gelernter, "Generative communication in linda," *ACM Trans. Program. Lang. Syst.*, vol. 7, no. 1, pp. 80–112, 1985.