International Journal of Computer Applications (0975 8887)
Volume 78 - No. 4, September 2013

Multi-Agent System Design for Synchronizing Software Components Communication : Application on Orchestration in Complex SOA Architecture

Smail Tigani
RITM Laboratory, High School of Technology
National and High School of Electricity and Mechanics
sma.tigani@gmail.com
Casablanca, Morocco

Mouhamed Ouzzif
RITM Laboratory, High School of Technology
National and High School of Electricity and Mechanics
ouzzif@gmail.com
Casablanca, Morocco

ABSTRACT

This paper proposes a new optimal method for synchronizing flux orchestration services. By orchestration, we mean the process of communicating different applications in a collaborating context, generally in a complex SOA Architecture. Current way for resolving this problem is done manually by developers them selves, witch increases economical cost and programming complexity. Traditional method consumes physical resources and makes the software maintenance harder than ever because functional and non functional aspects are not separated. The proposed method is a multi-agent system composed by a set of slave agents and a master agent. A slave agent is an event's listener and an alert sender to the master agent. This one makes adequate reaction based on a primary simple configuration.

Keywords:

Multi-Agent System, Synchronization, Orchestration, SOA Architecture, Web Services, Aspect Oriented Programming "AOP".

1. INTRODUCTION

With the expansion of information technologies and the development of communication protocols, companies needs more than one application for their departments management. Those applications have to exchange information between them, let's give an example : When a human resources manager allows an employee to get a holiday during a given period, the Human Resources Software must send a message to the Project Management Software to make it update and do not consider the employee present during this period.

Service Oriented Architecture is a set of applications interacting between them, and each one offers services to others. Developers have to spend time and effort by adding manually some Web Service calls to communicate with other components, this decreases the application performance and make it's maintenance harder due to coding complexity.

This paper proposes an architecture of a new system allowing developers to communicate remote software components easily by using just java annotations. This solution optimizes also some performances aspects of the information system.

This paper is organized as follows : the next section presents the global context by introducing some definitions, while seconde section defines the problem and underlines motivations to think about a new solution. Architecture is reported in the third section. Section 4 introduces some mathematical aspects to demonstrate the proposed approach value-added. Finally, conclusion and some perspectives are presented in the last section.

2. SOLUTION CONTEXT

2.1 WEB SERVICES

According to [1], a web service is an interface that offers a list of operations witch are accessible using a network. Web services description is done by a formal XML standard, named WSDL "Web Service Description Language". It contains all necessary information to interact with the service and other informations like message formats, transport protocols and location... The interface hides the implementation details of the service and that make it independent of the hardware or software platform on which it is implemented and also independent of the programming language and technology in which it is based. This allows applications based on web Services to cooperate and that increases distributed applications interoperability.

2.2 ABOUT SOA

By reading [2], SOA is some principles, patterns and best practices that used in software components developing. SOA means "Service Oriented Architecture" and speaks about services and only abstract interfaces are outside accessible. Implementation details are hidden and consumers are unaware of them, additionally, there are others either indifferent however interfaces are neutral [1] of programming languages and various technologies and platforms.

2.3 WHAT IS SOAP ?

SOAP is a protocol used to assure interoperability between organizations and business applications needing interaction. It is an HTTP POST request with an XML structure wrapped in HTTP Playload field, some web services types are based on SOAP protocol. Programs behaving like a web navigator can send and receive SOAP packets[3].

2.4 SOA AND WEB SERVICES

Time has proven that web services technology is an efficient way to implement service oriented architectures [4] interconnecting distributed software components and construct by this a collaborative context. Web service's platforms and technologies independence makes them a viable technology to attain SOA since they are based on industry standards like WSDL, SOAP and XML.

2.5 AGENT TECHNOLOGY

An agent is a autonomous entity supervising its environment with its sensors, it can behaves with its effectors to attain an objective or the list of objectives designed for. Different agents have different influence level in the sense that they have influence on specific parts of the environment[5].

2.6 MULTI AGENT SYSTEMS

An agent have to the ability to communicates with other agents, that create an agents network and all the system is called Multi Agent System "MAS". They are applied in different fields like software engineering, mechanical systems ... According to [6], a distributed watershed systems optimization is done using a system based on agent technology.

3. SYSTEM DESIGN

3.1 PROBLEM DEFINITION

Web applications are generally developed using the MVC Design Pattern, this one is a multi-layer architecture grouping the application in tree main layers : Model, View and Controller. The Model contains business model "BM" which is the object relational mapping, however, the View contains the human machine interface "IHM" and the Controller contains business services "BS".

For more interaction with other applications, we can imagine a web services sub-layer to communicate the software with other remote components.

Developers must code manually the interaction part using Web Services. First, this needs an intelligence level and time. Seconde, web services instances needs memory space and the call consumes time. The three previous points are considered the main drawbacks of the traditional method.

3.2 SYSTEM ARCHITECTURE

The proposed solution is working with a multi-agent system to avoid the three drawbacks. Slave agent will be an event listener, an event can be before or after method running. When the slave agent senses a given event, it sends a message containing required parameters to the master agent who extracts, from its configuration file, the adequate behavior : calling an other web service or a business process...

The next diagram contains different components and interconnection links :

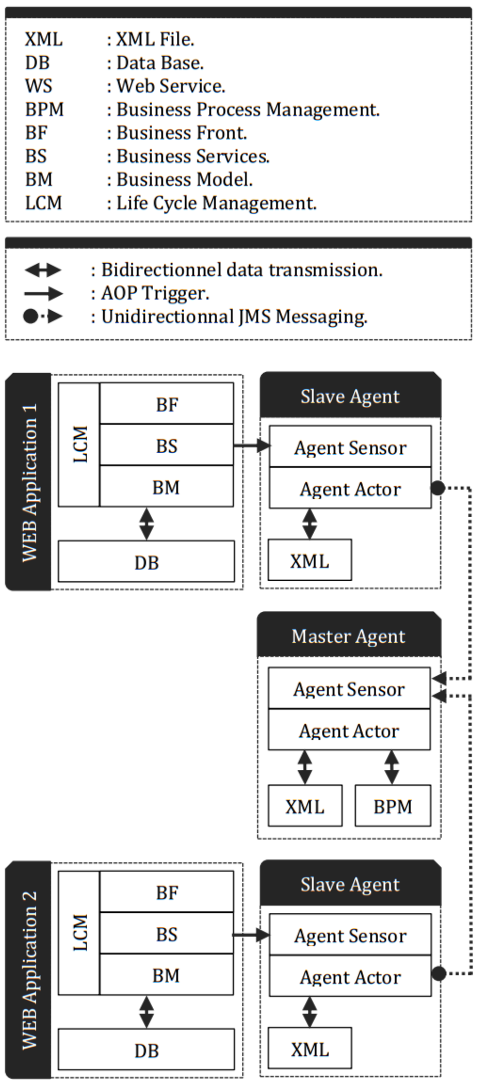

Fig. 1: Global system architecture

3.3 CONFIGURATION FILES

The agent needs some informations to make the right decision, it can extract them from an XML file completed by the developer with simple tags.

3.3.1 SLAVE AGENT XML FILE. Slave agent need next informations : event names list, Java method associated with the event and the channel Web Service URI of the master agent. This one is the UNIQUE Web Service created by the system and use it as a transmission channel for all events during the session.

3.3.2 MASTER AGENT XML FILE. Master agent need to know the associated operation for each received event.

3.4 HOW IT WORKS

Let's call "MA" the master agent and "SA" the slave agent. The following steps shows how the system works :

(1) A user send a request and that runs a method.
(2) The SA extracts events list.
(3) The SA detects the event.
(4) The SA extracts the method's parameters.
(5) The SA gets the message destination.
(6) The SA sends the event and parameters.
(7) The MA receives the message.
(8) The MA gets behavior list and parameters.
(9) The MA calls the adequate operation.

4. SYSTEM MODELLING

4.1 INTRODUCTION

In the current section, we focus on the mathematical demonstration of the efficiency of the proposed solution. First, we start by making some definitions of different variables and suppositions. Second, we will construct two models representing total storage size taken by Java Web Services Objets due to JVM instances and the response time. Finally, we will figure out the difference aspects between the proposed approach and current technology.

4.2 MAIN DEFINITIONS

Let's have n queries made by users to an application. A query can be the request of a list of products, or adding new informations in a database... Technically, this query involves one process witch is implemented by a Java Method or Function for example. Let's call p_i the i^{eme} process who needs to transmit information flux to other applications. It's not necessary that one process updates all applications in the system, this is why we suppose the function $N(p_i)$ as the number of applications concerned by the the update.

When a process calls a Web Service, the process have to instantiate the interface of the called Web Service before use. This operation, made by the JVM, means that the server memory will decrease by the Web Service object size. Let's call s_{ij} the size of the j^{eme} Web Service object called by the i^{eme} process p_i and t_{ij} the elapsed time when the j^{eme} Web Service call has been finished.

A process p_i it self needs a memory space and a time of running, let's call the two parameters $s(p_i)$ for the size and $t(p_i)$ for the time.

4.3 MEMORY SPACE MODEL

According to the suppositions in the previous section, we can build some performance indicators witch can help us to show the value-added of the proposed approach.

Let start by modelling the existing way to make orchestration. The memory space for n processes is given by $\sum_{i=1}^{n} s(p_i)$ and each one have $N(p_i)$ Web Service calls, so the total memory including Web Services calls is represented by the following suit :

$$S_n = \sum_{i=1}^{n} \sum_{j=1}^{N(p_i)} s_{ij} + s(p_i)$$

Let's define the same parameters for our approach, the difference is there is no Web Services calls. We have just a small size for creating <u>one</u> instance of the agent java class, it's a singleton, Let's call it S_{SA} : "Size of Slave Agent". Finally, the total size is given by the following suit :

$$S_n^* = S_{SA} + \sum_{i=1}^{n} s(p_i)$$

Now let's study the suit $\Delta_n^S = S_n - S_n^*$ witch give us the best solution according to it's convergence speed. The result is given by the next equation :

$$\Delta_n^S = -S_{SA} + \sum_{i=1}^{n} \sum_{j=1}^{N(p_i)} s_{ij}$$

It's evident that $\lim_{n \to \infty} \Delta_n^S = \infty$ because $s_{ij} > 0$. That means that $S_n \to \infty$ more quickly than S_n^*. This result means that the server memory will be saturated quickly if we use the traditional method. Finally we conclude that our approach represented by S_n^* makes more economics in memory space.

4.4 RESPONSE TIME MODEL

By the same way, total elapsed time after n process running including Web Services calls is given by the next suit :

$$T_n = \sum_{i=1}^{n} \sum_{j=1}^{N(p_i)} t_{ij} + t(p_i)$$

We consider T_{SA} the time to create the object and running the main method, this time is small and this operation is done just one time. The total elapsed time in this case is obtained by the next suit :

$$T_n^* = T_{SA} + \sum_{i=1}^{n} t(p_i)$$

Now let's study the suit $\Delta_n^T = T_n - T_n^*$ witch give us the best solution according to it's convergence speed. The result is given by the following equation :

$$\Delta_n^T = -T_{SA} + \sum_{i=1}^{n} \sum_{j=1}^{N(p_i)} t_{ij}$$

We have $\lim_{n \to \infty} \Delta_n^T = \infty$, so we can conclude that the response time of the application will be higher if we use the traditional method. Finally we can say that our approach represented by T_n^* optimises the performance of the application.

5. CONCLUSION

In this paper, we figure out some weaknesses of the traditional method of developing orchestration services. On the one hand, developers them selves have to make every thing manually, this is complex to do and makes the software maintenance harder. On the other hand, some performance aspects are not optimized us we have seen before.

To make orchestration as simple as possible, this work proposes a distributed system architecture composed of slave and master agents to replace the manual human works. With this approach, the application container have no need to create web services instances and wait for the response. This is more economical in space and time.

As a perspective, the implementation of this architecture will be with Java Technology. The slave and master agent will be developed with the EJB3.x API and consumes an XML file containing the configuration, this will be with the JAXB API. Follows message exchages class with JAX-WS API[7], and the ApsectJ API[8] to detect after and before method running events.

6. REFERENCES

[1] H. Kreger, *Web Services Conceptual Architecture*, pp 6, 2001.

[2] C. A Binildas, *Service Oriented Java Business Integration*, pp 32, PACKT, 2008.

[3] P. Busby, *A Simple Way of Importing from a REST Web Service into SAS in Three Lines of Code*, pp 1 & 2, 2012.

[4] C. A Binildas, *Service Oriented Java Business Integration*, pp 33, PACKT, 2008.

[5] S. MAALAL, M. ADDOU, *A new approach of designing Multi-Agent Systems*, pp 148, International Journal of Advanced Computer Science and Applications, Vol. 2, No. 11, 2011.

[6] M. Giuliani, A. Castelletti, F. Amigoni, X. Cai *Multi-Agent Systems optimization for distributed watershed management*, pp 2, International Environmental Modelling and Software Society, 2012.

[7] M. Kalin, *Java Web Services : Up and Running*, O'REILLY, 2008.

[8] R. LADDAD, *AspectJ in action*, MANNING.