TOWARDS THE DYNAMIC ADAPTABILITY OF SOA

Mehdi Ben Hmida[1], Céline Boutrous Saab[1], Serge Haddad[1]
Valérie Monfort[1,2] and Ricardo Tomaz Ferraz[2]

[1] *LAMSADE-CNRS, Université Paris-Dauphine, Place du Maréchal de Lattre Tassigny, Paris Cedex 16, France*

[2] *CRI, Université Paris 1 Sorbonne, 90 rue de Tolbiac, 75013 Paris, France*

Keywords: Service Oriented Architecture (SOA), Web Services (WS), Aspect Oriented Programming (AOP), Process Algebra (PA), Dynamic Adaptability.

Abstract: Service Oriented Architectures (SOA) aim to give methodological and technical answers to achieve interoperabilty and loose coupling between heterogeneous Information Systems (IS). Currently, Web Services are the fitted technical solution to implement such architectures. However, both Web Services providers and clients are faced to some important difficulties to dynamically change their behaviours. From one side, Web Services providers have no mean to dynamically adapt an existing Web Service to business requirements changes. From the other side, Web Services clients have no way to dynamically adapt themselves to the service changing in order to avoid execution failures. In this paper, we show how to achieve a dynamic adaptable SOA by using the Aspect Oriented Programming (AOP) paradigm and Process Algebra (PA) formalism. We extend our previous works to dynamically modify BPEL processes and to handle client-server communications issues. Then, we use a process algebra formalism to specify a change-prone BPEL process and demonstrate how to generate a client which dynamically adapt its behaviour to the service changes. We also present the *Aspect Service Weaver* (ASW) prototype which implements our approach.

1 INTRODUCTION

Web Services (WS) are "self contained, self-describing modular applications that can be published, located, and invoked across the Web" (Tidwell, 2000). They are based on a set of XML (Bray et al., 2004) standards to make it more portable than previous middleware technologies. SOA-based applications are usually composed of simple WSs that are offered by different providers. The *Business Process Execution Language for Web Services* (*BPEL*) has been introduced for this purpose and becomes a standard (Andrews et al., 2003). BPEL supports two different types of business processes: *Executable processes* executed on a BPEL engine and *Abstract business processes* specifying the interaction protocol with the client.

Actually, SOA applications are faced to some important limitations concerning their adaptability to business requirements changes. These limitations affect both WSs providers and consumers.

First, the services providers have no mean to dynamically change their WSs implementation or composition. They need to undeploy the service, recodify the business logic and redeploy it again. This scenario is deficient in an industrial context with a high Time-to-Market constraints and strong competitiveness between companies. Second, if a change on the service description (WSDL) or in the interaction protocol (abstract BPEL) is done (without new publication), the service consumers could not more interact with the modified service and this will lead to execution failures

Regarding the above limitations, we identified two requirements that WS technology has to handle:

1. Service provider needs to dynamically change the behaviour of an already existing service to adapt it to the new applications requirements.
2. Service client needs to dynamically adapt itself to the service changing to avoid execution failures.

In our previous works, we addressed dynamic ser-

Ben Hmida M., Boutrous Saab C., Haddad S., Monfort V. and Tomaz Ferraz R. (2007).
TOWARDS THE DYNAMIC ADAPTABILITY OF SOA.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - ISAS*, pages 474-479
DOI: 10.5220/0002387504740479

vice adaptability and client interaction issues. We proposed an Aspect Oriented Programming (AOP) (Kiczales et al., 1997) approach which aims to change elementary WSs at runtime (Hmida et al., 2006; Tomaz et al., 2006). We also proposed a Process Algebra (PA) approach which solves the interaction problem between BPEL processes and its clients, by formally specifying the interaction protocol (abstract BPEL) and automatically generating a correct client (Haddad et al., 2006). In this paper, we extend these works in order to reach the objectives previously discussed.

This paper is organized as follows: Section 2 briefly presents our previous AOP approach for elementary WSs, then shows its extension to support BPEL processes and to handle interaction issues. We also present the architecture of our Aspect Service Weaver (ASW) tool which integrates these concepts. Section 3 presents the process algebra formalism which supports change-prone BPEL processes. This formalism leads us to generate clients that adapt themselves to the service changes. Section 4 discusses related works. We conclude and present some future works in section 5.

2 DYNAMIC SERVICE ADAPTABILITY

2.1 Aspect Oriented Programming

Many researches (Charfi and Mezini, 2004; Courbis and Finkelstein, 2005; Verheecke et al., 2003) consider Aspect Oriented Programming (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. AOP concepts were formulated by Chris Maeda and Gregor Kiczales (Kiczales et al., 1997). 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 behaviour. The weaving can occur at compile time (modifying the compiler), load time (modifying the class loader) or runtime (modifying the interpreter).

2.2 Applying Aop to Web Services : The Aspect Service Weaver (ASW)

In our previous approach, we developed an AOP-based tool named *Aspect Service Weaver* (*ASW*) (Hmida et al., 2006; Tomaz et al., 2006). The ASW intercepts the SOAP messages between a client and an elementary WS , then verifies during the interaction if there is a new behaviour introduced (*advice services*). We use the AOP weaving time to add the new behaviour (*before*, *around* or *after* an activity execution). The advice services are elementary WSs whose references are registered in a file called "*aspect services file descriptor*". The pointcut language is based on XPath (Clark and DeRose, 1999). XPath queries are applied on the service description (WSDL) to select the set of methods on which the advice services are inserted.

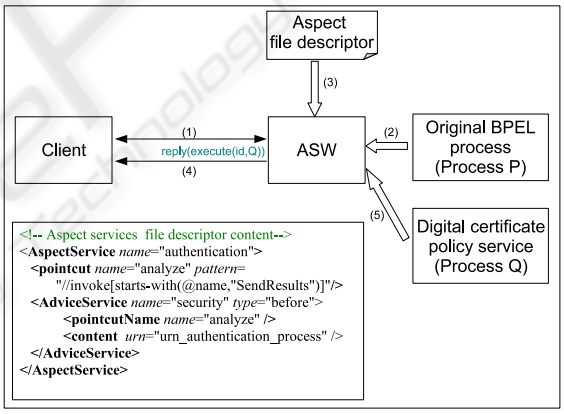

Figure 1: Interaction schema for the insertion of a security policy.

We extend this approach to BPEL processes. The *ASW* controls the BPEL process execution instead of intercepting SOAP messages. It is integrated in the BPEL engine in order to interpret the BPEL process and apply the aspect services. It verifies before the execution of each BPEL activity if some *Aspect service* has ti be inserted. Then, it executes the corresponding *advice service*. We also add a new functionality to the *ASW*. The tool dynamically generates messages called *execute* messages, encapsulating the identifier and the interaction protocol of the *advice service*. These messages are sent to the client to advertise it about a new behaviour inserted at runtime. This message is necessary since the new behaviour can require new information exchange involving messages not expected by

the client, leading to execution failures. At the client implementation, the developer has to handle this type of message: it has to extract the interaction protocol of the *advice service* and integrate it in its behaviour. This part is detailed in the next section.

Considering a scenario where a service developer wants to change dynamically a Kerberos token security policy by a digital certificate ones. He can develop an aspect service called "authentication" and specify, in the "aspect services file descriptor", that before the invocation of the authenticated methods in the base BPEL process, the engine must invoke a digital-certificate authentication instead of the Kerboros token security. For instance, The "aspect services file descriptor" in Figure 1 indicates to the engine that when the methods whose names match with the XPath expression "//invoke[starts-with(@name,"SendResult")]" were invoked (equivalent to the invocation of methods whose names match the regular expression "SendResult*"), the engine must invoke before the advice service digital-certificate.

This way, during the normal process execution (step 2 in Figure 1) , the ASW looks in the "aspect services file descriptor" (step 3) for *aspect services* applied to the current activity. It finds (for example) a joinpoint matching between a "sendResult" method invocation and the aspect "authentication". Then, it generates the *execute* message encapsulating the interaction protocol of the *advice service* identified by *id* and sends it to the client (step 4). After that, the *ASW* begins the execution of the *advice service* (step 5). When its execution terminates, the *ASW* continues the execution of the base process.

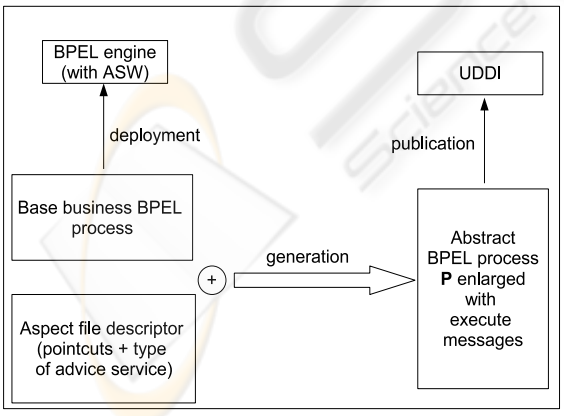

Figure 2: The extended abstract BPEL process.

The change-prone BPEL process interaction protocol is described by an extended abstract BPEL process which integrates the sending of *execute* messages. The extended interaction protocol is generated from the base BPEL process and the *aspect service file descriptor* based on the defined pointcuts and the type of advices (before, after or around) (figure 2).

The generation process performs transformations on the base BPEL process syntactic tree. It inserts the action of sending *execute* messages in the selected joinpoints depending on the kind of the *advice service*. The figure 3 shows the transformations made on the abstract base process $sequence(receive(ResReq), switch(reply(ResResp), reply(error))$ which receives a *ResReq* message then replies by a *ResResp* or *error* message depending on an internal action (the switch process). In the case of an around *service advice* (figure 3.d), the specified *joinpoint* is replaced by the $reply(execute(id))$ message because we consider that the *advice service* can encapsulate the joinpoint.

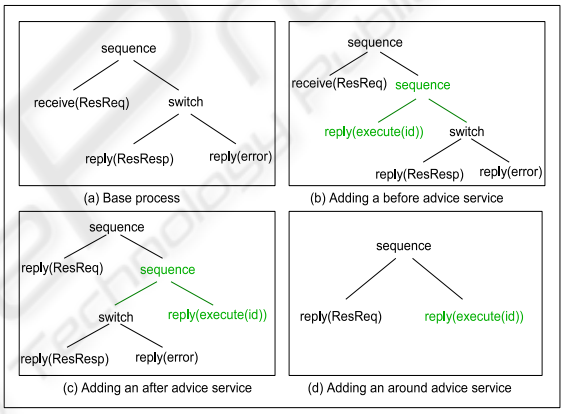

Figure 3: transformations on the syntactic BPEL process tree.

In the extended abstract BPEL process, the *execute* message contains only the identifier of the *advice service* (id). The interaction protocol corresponding to that *id* is sent to the client at runtime.

3 DYNAMIC CLIENT ADAPTABILITY

BPEL provides a set of operators describing in a modular way the observable behaviour of an abstract process. As shown in (Staab et al., 2003), this kind of process description is close to the process algebra paradigm illustrated for instance by CCS (Milner, 1995).

However, time is explicitly present in some of the BPEL constructors and thus the standard process algebra semantics are inappropriate for the description

of such process. Thus, we defined a new process algebra semantics that associates a timed automaton (TA) (Alur and Dill, 1994) with an abstract process (Haddad et al., 2006). The theorical developments follow these steps: associating operational rules with each abstract BPEL construct, defining an interaction relation which formalizes the concept of a correct interaction between two communicating systems (the client and the WS), and designing an algorithm that generates a client automaton which is in an interaction relation with the WS.

The client automaton is interpreted by our generic client interpreter (figure 4). Our client downloads the abstract BPEL process from an UDDI registry and generates its corresponding TA. Then, based on the TA of the service and the interaction relation, it generates the client TA if the service is not ambiguous. Finally, it executes the client TA and displays graphical interfaces allowing to the human user to enter the messages parameters.

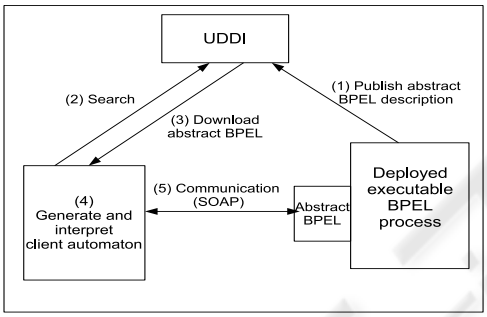

Figure 4: Generic client interpreter.

3.1 The Dynamic Client Interpreter

In order to communicate with change-prone BPEL processes, we extend the previous client interpreter. The new client has to achieve the following tasks:

1. When the client receives an $execute(id)$ message, it has to extract the *advice service* interaction protocol (identified by *id*) and generates its corresponding server and client TA.
2. It simultaneously executes the client TAs of the main process and its *advices clients* TA.
3. It makes synchronisation between the main client TA and the advices clients TA on the termination of services advices execution.

Furthermore, the generation module of the dynamic client interpreter also integrates new operational rules for the sending and receiving processes in order to handle the $execute(id)$ messages.

3.2 Formalisation Steps

In order to formalize BPEL as dense timed process algebra, we have to define the actions (alphabet) of the process algebra. The possible actions are message receiving ($?m$) and sending ($!m$), internal actions (τ) (not observable from the client side), raise of exceptions ($e \in E$), expiration of timeout (to) and the termination of the process ($\surd$). We distinguish three kinds of actions: the immediate actions corresponding to a logical transition $(\tau, e, \surd)$, the asynchronous actions where an unknown amount of time elapses before the occurrence of actions $(?m, !m)$ and the synchronous actions (to) which occur after a fixed delay.

Now, we present some operational rules and precisely the new rules for the sending and receiving processes. To see all rules and in particular the handling of clocks in TA, the reader is invited to refer to (Haddad et al., 2006).

For example, the *empty* process which represents the process that does nothing can only terminate by executing the $\surd$ action (0 is the *null* process).

$$empty \xrightarrow{\surd} 0 \tag{1}$$

For the sending and receiving processes, we define the following rules.

$$\forall m \neq execute$$
$$*o[m] \xrightarrow{*m} empty \qquad avec \ * \in \{?, !\} \tag{2}$$

$$!o[m] \xrightarrow{!execute(id)} WaitAdvice(id) \tag{3}$$

$$WaitAdvice(id) \xrightarrow{id.\surd} empty \tag{4}$$

Rule 2 states that the process $?o[m]$ (resp. $!o[m]$) which corresponds to the reception of a message of type m (resp. sending of message of type m) executes the action $?m$ (resp. the action $!m$) which corresponds to the message reception action (resp. the message sending action) and becomes the *empty* process. In the case of sending an *execute* message, the automaton evolves to an intermediary state named $WaitAdvice(id)$ (rule 3). $WaitAdvice(id)$ waits for the termination of the *advice service* identified by *id*. When *advice service id* terminates, $WaitAdvice(id)$ state executes $id.\surd$ and becomes *empty* process (rule 4).

The sequential process $P;Q$ (P and Q are BPEL processes) corresponds to the execution of the process P followed by the execution of the process Q. It becomes the process $P';Q$ if the process P executes an action a different from termination action and becomes P'. If P terminates and Q can execute an action

a and becomes Q', the process $P;Q$ executes the action a then becomes the process Q'.

$$\forall a \neq \surd \frac{P \xrightarrow{a} P'}{P;Q \xrightarrow{a} P';Q} \quad (5)$$

$$\frac{P \xrightarrow{\surd} \text{ and } Q \xrightarrow{a} Q'}{P;Q \xrightarrow{a} Q'} \quad (6)$$

Finally, the $switch\{P_i\}_{i\in I}$ process evaluates an internal condition represented by τ then becomes the process P_i.

$$\forall\, i \in I,\ switch\{P_i\}_{i\in I} \xrightarrow{\tau} P_i \quad (7)$$

3.3 Execution Scenario

Considering the abstract BPEL process defined in section 2. If we want to add dynamically an authentication process before the *switch* process, the extended abstract BPEL process have to integrate a sending $execute(id)$ message process before the *switch* process.

$?o[ResReq];!execute(id);switch(!o[ResResp],!o[error])$

At the execution, our dynamic client interpreter downloads the extended abstract BPEL specification. Then, it generates the corresponding service TA based on the operational rules previously defined. Then, based on the service TA and the interaction relation, our client generates the client TA and begins its interpretation. Figure 5 shows the generation process.

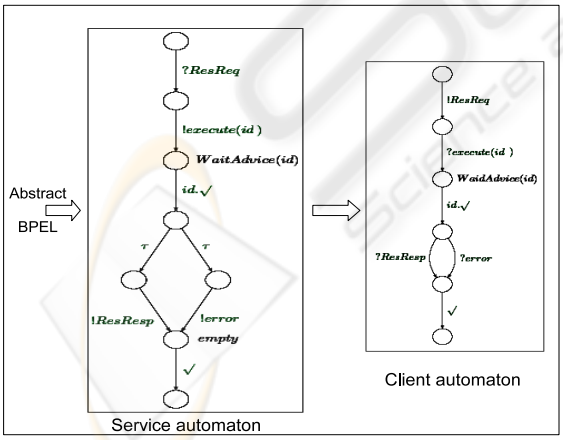

Figure 5: Adaptable service and client automata.

When our client receives an $execute(id)$ message, it extracts the abstract BPEL *advice service* process from the message. In our example, the *advice service* is an authentication process which abstract BPEL specification is $!o[authDataRequest]$; $?o[authDataResp]$;$P1$. This process sends an authentication data request to the client asking for authentication data, receives these data then performs some actions to authenticate the user. Our client generates the corresponding advice client automaton, associates with the received *id* and begins its execution (Figure 6, states in grey represents the current execution step).

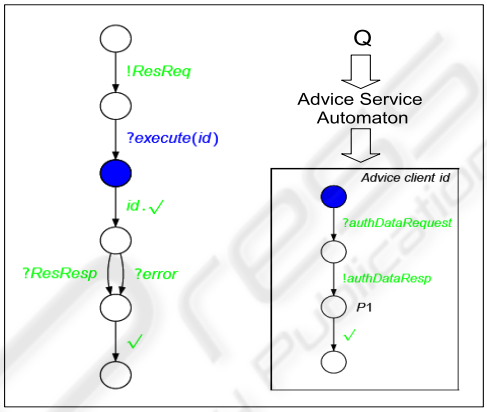

Figure 6: Receiving an $execute(id,Q)$ message.

When the *advice client id* terminates, our client makes synchronisation with the main client automaton. it deletes the *advice client*, performs the $id.\surd$ action and continues the execution of the main client automaton (figure 7).

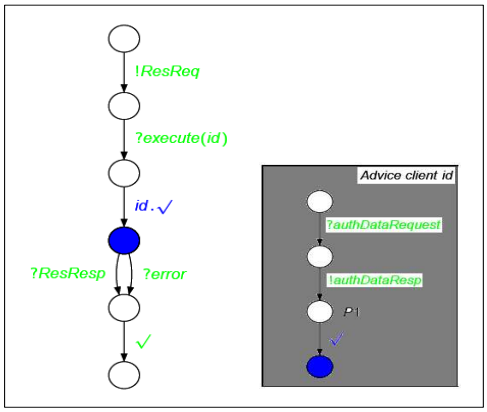

Figure 7: Termination of the advice client *id*.

4 RELATED WORKS

In (Charfi and Mezini, 2004) and (Courbis and Finkelstein, 2005), the authors define specific AOP lan-

guages to add dynamically new behaviours to BPEL processes. But, neither of these approaches address the client interaction issue. The client has no mean to handle the interactions that can be added or modified during the process execution.

The Web Service Management Layer (WSML) (Verheecke et al., 2003) is an AOP-based platform for WSs that allows a more loosely coupling between the client and the server sides. WSML handles the dynamic integration of new WSs in client applications to solve client execution problems. WSML dynamically discover WSs based on matching criteria such as: method signature, interaction protocol or quality of service (QOS) matching. In a complementary way, our work proposes to adapt a client to a modified WS.

Some proposals have emerged recently to abstractly describe WSs, most of them are grounded on transition system models (Labelled Transition Systems, Petri nets, etc.) (Hamadi and Benatallah, 2003; Fu et al., 2004; Ferrara, 2004). These works propose to formally specify composite WSs and handle the verification and the automatic composition issues. But, neither of these works propose to formalize the dynamics of SOA architectures and to handle runtime interaction changes.

5 CONCLUSION AND FUTURE WORKS

In this paper, we proposed a solution based on AOP and PA to handle dynamic changes in the WS context. We extended our previous AOP approach to support BPEL processes and to handle interaction issues. We also use process algebra formalism to specify change-prone BPEL processes and generate dynamic clients.

As future works, we want to extend the work to take into account the client execution context. We also want to formally handle the aspects interactions issue (aspects applied at the same joinpoint). Finally, we plane to improve the current ASW prototype as proof-of-concepts.

REFERENCES

Alur, R. and Dill, D. L. (1994). A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235.

Andrews, T. et al. (2003). *Business Process Execution Language for Web Services*. 2nd public draft release, Version 1.1, http://www.ibm.com/developerworks/webservices/library/ws-bpel/.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., and Yergeau, F. (2004). Extensible markup language(xml) 1.0, http://www.w3.org/xml/.

Charfi, A. and Mezini, M. (2004). Aspect-oriented web service composition with AO4BPEL. In *Proceedings of the 2nd European Conference on Web Services (ECOWS)*, volume 3250 of *LNCS*, pages 168–182. Springer.

Clark, J. and DeRose, S. (1999). Xml path language (xpath) ver. 1.0, http://www.w3.org/tr/xpath.

Courbis, C. and Finkelstein, A. (2005). Weaving aspects into web service orchestrations. In *ICWS '05: Proceedings of the IEEE International Conference on Web Services (ICWS'05)*, pages 219–226, Washington, DC, USA. IEEE Computer Society.

Ferrara, A. (2004). Web services: a process algebra approach. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 242–251, New York, NY, USA. ACM Press.

Fu, X., Bultan, T., and Su, J. (2004). Analysis of interacting bpel web services. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 621–630, New York, NY, USA. ACM Press.

Haddad, S., Moreaux, P., and Rampacek, S. (2006). Client synthesis for web services by way of a timed semantics. In *Proceedings of the 8th Int. Conf. on Enterprise Information Systems (ICEIS06)*, pages 19–26.

Hamadi, R. and Benatallah, B. (2003). A petri net-based model for web service composition. In *ADC '03: Proceedings of the 14th Australasian database conference*, pages 191–200, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

Hmida, M. B., Tomaz, R. F., and Monfort, V. (2006). Applying aop concepts to increase web services flexibility. *Journal of Digital Information Management (JDIM)*, 4(1):37–43.

Kiczales, G., Lamping, J., Maeda, C., and Lopes, C. (1997). Aspect-oriented programming. In *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York.

Milner, R. (1995). *Communication and concurrency*. Prentice Hall International (UK) Ltd., Hertfordshire, UK.

Staab, S., van der Aalst, W., and Benjamins, V. R. (2003). Web services: been there, done that? *IEEE Intelligent Systems [see also IEEE Intelligent Systems and Their Applications]*, 18(1):72–85.

Tidwell, D. (2000). Web services: The web's next revolution, http://whitepapers.techrepublic.com.com/.

Tomaz, R. F., Hmida, M. B., and Monfort, V. (2006). Concrete solutions for web services adaptability using policies and aspects. *The International Journal of Cooperative Information Systems (IJCIS)*, 15(3):415–438.

Verheecke, B., Cibrán, M., and Jonckers, V. (2003). AOP for Dynamic Configuration and Management of Web Services. In *Proceedings of the International Conference on Web Services Europe 2003*, volume 2853, pages 137–151. Springer.