ENHANCING ENTERPRISE COLLABORATION USING CONTEXT-AWARE SERVICE BASED ON ASPECTS

Khouloud Boukadi[α], Chirine Ghedira[β] and Lucien Vincent[α]
[α]*Division for Industrial Engineering and Computer Sciences, ENSM, Saint-Etienne, France*
[β] *LIRIS Laboratory, Claude Bernard Lyon 1 University, Lyon, France*

Keywords: Web service, BPEL, service adaptation, context-aware, Aspect Oriented Programming.

Abstract: In fast changing markets, dynamic collaboration ability involves establishing and "enacting" business relationships in an adaptive way taking into account context changes. This relies on using adaptable and flexible IT platforms. Service orientation can address this challenge. Accordingly, collaborative processes can be implemented as a composition of a set of services. However, combining "directly" elementary IT services is a hard task and presents risks in both service provider and user sides. In this paper we present a high-level structure called Service Domain which orchestrates a set a of related IT services based on BPEL specifications. To ensure the Service Domain adaptability to context changes our approach aims to prove the benefits of bringing Aspect Oriented Programming.

1 INTRODUCTION

Continuing globalization and changing customer needs are forcing enterprises to rethink and restructure their business models and organizational structures. To stay competitive, enterprises have to be increasingly efficient, flexible, and innovative. They will focus more on core competencies and outsource other activities to dynamically selected partners to deliver the best possible customer value and the shortest time-to-market.

IT systems play a crucial role in enterprise. They must allow the enterprise to respond to changes which occur in a timely, dynamic, and reliable manner without compromising organizational flexibility. This brings into focus the role of defining and implementing flexible business processes supported by flexible IT systems, which allow enterprises to collaborate with partners in dynamic and flexible way. Flexible IT systems are those that are malleable enough to deal with context changes in an unstable environment (Byrd and Turner, 2001).

The emergence of the service-oriented computing (SOC) paradigm and Web services technology, in particular, has aroused enormous interest in service-oriented architecture (SOA). Based on the service orientation, existing IT infrastructure can be bundled and offered as Web services with standardized and well defined interfaces. We call Web services arising from applying this process enterprise IT Web services or shortly IT services.

1.1 Enterprise IT Services

Enterprise publishes IT services in order to be used inside and outside the enterprise. IT services can be combined and recombined into different solutions and scenarios, as determined by business needs. IT services promote business processes by composing individual IT services to represent complex processes, which can even span multiple organizations. However, transforming enterprise IT infrastructure into a large set of published IT services with different granularity levels has a number of drawbacks. Firstly, it may imply that an enterprise has to expose service elements which are, in isolation meaningless, to the outside world. Secondly, service consumer will undertake several and low-level service combinations and this will overburden its task, thereby decreasing the added value of service provisioning. Thirdly, in this form a service consumer can compose a process which has no sense to the service provider. To overcome these limits, we believe that an enterprise must re-organize its IT services and presents its functionalities through a high-level service. In this work, we

Boukadi K., Ghedira C. and Vincent L. (2008).
ENHANCING ENTERPRISE COLLABORATION USING CONTEXT-AWARE SERVICE BASED ON ASPECTS.
In *Proceedings of the Tenth International Conference on Enterprise Information Systems - SAIC*, pages 29-36
DOI: 10.5220/0001714200290036

develop a high level structure called *Service Domain* (SD) which represents a combination of related IT services as a single logical service. Service Domain orchestrates a set of IT services in order to give a high level functionality and a comprehensible external view to the end user. Service Domain will be published as a Web service, thus hiding the complexity of publishing, selecting and combining fine grained IT services.

Furthermore, in order to satisfy enterprise adaptability to context changes, Service Domain must be more than a functionality provided through the Web. Indeed, it must have the capacity to adapt own its own behaviour by comporting appropriately to accommodate the situation in which it evolves. To respond to this characteristic, Service Domain has to assess its current capabilities, the ongoing commitments, and the surrounding environment. As a result Service Domain must be context-aware.

1.2 Contribution and Paper Organization

Our aim in this paper is to present the concept and the architecture of the Service Domain which orchestrates a set of related IT services based on the Business Process Execution Language (BPEL[1]) specifications (Andrews and Curbera, 2003). Service Domain has in addition the characteristic of being context-aware. Context awareness is guaranteed by enhancing BPEL execution using Aspect Oriented Programming (AOSD, 2007). Indeed, AOP enable crosscutting concerns which is crucial for managing context information separately from the business logic implemented in the BPEL process

The rest of the paper is organized as follows. First, in section 2, we present the SD architecture. Then, in Section 3, we expose our context categorization and we highlight the limits of BPEL to address adaptability to the context changes. In section 4, we introduce the Aspect Oriented Programming and how we use it to enhance BPEL adaptability. In addition, we expose a running example and implementations. Finally, Section 5 details some related work.

[1] BPEL is an XML-based language designed to enable task-sharing for a distributed computing even across multiple organizations using a combination of Web services.

2 SERVICE DOMAIN CONCEPT

Service Domain (SD) is a high level structure used to manage a large number of enterprise IT services. In fact, a set of related IT services are gathered within one SD which will be empowered with suitable facilities to assist and enhance IT services uses. The SD enhances the Web service concept. In fact, its purpose is not to define new application programming interfaces (APIs) or new standards, but rather, to provide, based on existing IT services, a new higher-level structure that can hide complexities from service users, simplify deployment for service suppliers and provide self-managing capabilities. Service Domain is based on/uses Web service standards (i.e. WSDL, SOAP and UDDI).

The Service Domain will be used as major building block for implementing enterprises business processes, which will be represented as a composition of Service Domains that belong to different enterprises (see Figure 1).

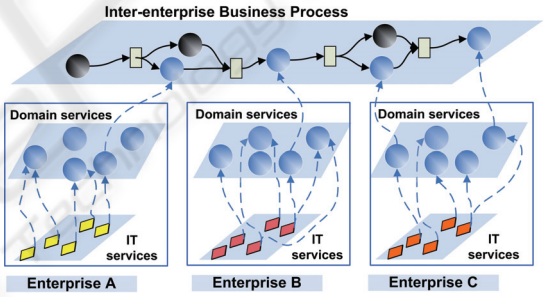

Figure 1: Inter-enterprise collaboration based on Service Domain.

The Service Domain orchestrates and manages several IT services as a single virtual service. It promotes a SOA solution which decreases the intricacy of providing business applications.

As an example of a Service Domain, we can consider a "logistic enterprise" which exposes a "Delivery Service Domain" (DSD) which constitutes on merchandise delivery service. DSD encapsulates five IT services: "Picking merchandise", "Verifying merchandise", "Putting merchandise in parcels", "Computing delivery price" and "Transporting merchandise". Keeping these IT service in one place facilitates manageability and avoid extra composition work in the client side as well as exposing non-significant services like "Verifying merchandise" in the enterprise side.

The Service Domain is implemented as a node consisting of an Entry Module, a Context Manager

Module (CMM), Service Orchestration Module (SOM) and finally an Aspect Activator Module (AAM) as presented in Figure 2.

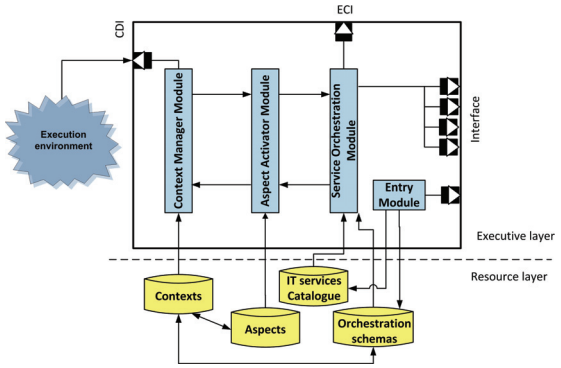

Figure 2: Service Domain architecture.

In this last Figure, three of these Modules provide external interfaces to the Service Domain node: Entry Module, Context Manager Module, and Service Orchestration Module. The Entry Module is based on Web service standard (SOAP) for receiving requests and returning responses. Aside from service requests from clients, the Entry Module also supports administrative operations for managing the Service Domain node. For example, an administrator can send a register command in order to add a new IT service with a given Service Domain by registering it into the corresponding IT service catalogue. The register command can also deal with a new orchestration schema which will be added in the orchestration schemas registry.

When the Entry Module receives an incoming request, it communicates with the orchestration schemas registry in order to select a suitable orchestration schema and identify the best IT service instances to fulfill the request. The selection of the orchestration schema and IT service instances, takes into account the context of the incoming request. Orchestration schemas with the set of IT service instances are delivered to the Service Orchestration Module (orchestration engine). The orchestration of different IT services belonging to one Service Domain is ensured using BPEL. The SOM presents an external interface called Execution Control Interface (ECI) which enables a user to obtain information regarding the state of execution of the SD internal process. This interface is very useful in case of external collaboration since it insures monitoring of the internal process execution. It presents a principal difference between our Service Domain and the traditional Web service. In fact, with the ECI interface, SD is based on the Glass box principles in contrast to the Web service which is based on the black box principles. Finally, the last external interface called Context Detection Interface (CDI) is used by the CMM to catch context information changes. Context detection is used to guarantee the SD adaptability. Adaptability of the SD is based on selecting and injecting the right Aspect according to the context change. To fulfil this requirement, SD uses the AAM to identify the suitable Aspect related to the context information and inject it in the BPEL process. This guarantees greater flexibility by quickly adapting the execution of the SD without stopping and redeploying it.

3 BPEL AND CONTEXT

The push towards context-aware, adaptive and on demand computing requires keeping Service Domain with suitable infrastructure which supports the delivery of adaptive services with varying functionalities.

Service Domain will be used in a context in which several factors call for dynamic execution evolution and changes (e.g., changes in the environment and unpredictable events). SD must meet with the requirements of customers' context changes as well as different service levels expectations. For instance, the Delivery Service Domain could advertise different behaviors by offering several delivery calculation methods depending on, for example, change in delivery location or time.

As SD uses BPEL to orchestrate a set of related IT services, its adaptability to context is closely related to the BPEL support of adaptability features.

In this section, we will expose the context paradigm, our context categorization and finally, we will present the shortage of BPEL to address the adaptability to context requirement.

3.1 Context Categories for the SD

Context appears in many disciplines as a meta-information which characterizes the specific situation of an entity, to describe a group of conceptual entities, and to partition a knowledge base into manageable sets or as a logical construct to facilitate reasoning services. Our definition of context follows the Dey's one (Dey et al., 2001) who says that a *context* is "*any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an*

application, including the user and the application themselves". The categorization of context is important for the development of context aware applications. Context includes implicit and explicit inputs. For example, user context can be deduced in an implicit way by the service provider such as in pervasive environment using physical or software sensors. Explicit context is determined explicitly by entities involved in the context. Bradely et al. depict that a variety of categorizations of context have also been proposed (Bradely and Dunlop, 2005). As a matter of fact, there are certain types of context which are, in practice, more used than others. These context categories are *location*, *identity*, *time,* and *activity*. Despite the various attempts to develop categorization for context, there is no generic context categorization. Relevant information differs from one domain to another and depends on their effective use (Mostafaoui and Mostafaoui, 2003) .

In this work, we propose a categorization of context using an OWL ontology (Bechhofer et al., 2004). Figure 3 depicts our context categorization ontology which is dynamic in the sense that new sub-categories may be added at any time. Each context definition belongs to a certain category which can be provider-related, customer-related and collaboration-related.

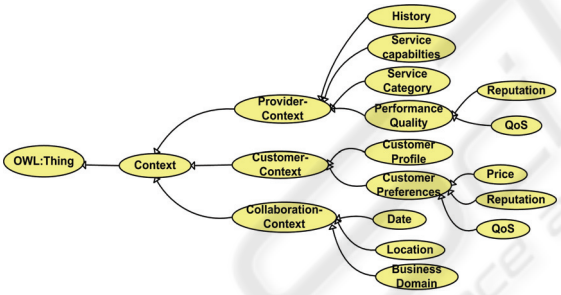

Figure 3: Ontology for categories of context.

In the following, we explain the different concepts which constitute our ontology based model for context categorization:

- A provider-related context deals with the conditions under which the providers can offer their Web services for externally. For example, the performance qualities include some metrics which measure the service quality such as time, cost, QoS, and reputation. These attributes model the competitive advantages that providers may have over each other.
- A customer-related context represents the set of available information and meta-data used by service providers to adapt their service. For example, a customer profile which represents a set of information characterizing the customer.
- A collaboration-related context represents the context of the business opportunity. We identify three sub-categories: location, time and business domain. The location and time represent the geographical location and the period of time within which the business opportunity should be accomplished.

3.2 BPEL Adaptation to Context

BPEL expose a static behavior which is inherited from the workflow management systems. This characteristic did not deal with evolutionary and runtime changes characterizing the execution environment (Charfi and Mezini, 2007). The only way is to stop the running process, modify the orchestration, and restart process execution. However, this is not a viable solution, especially for long-running and collaborative processes.

Actually, BPEL is silent in regards to the specification and handling of crosscutting concerns like context information. Then, it is difficult to define, modularize and manage context-sensitive behaviors. Traditionally, the implementation of adaptability extensions in BPEL gets scattered and tangled with the core functional logic. This in turn negatively impacts the system adaptability and scalability. Addressing these limitations, calls for developing new principles for building such SD, and for extending BPEL capabilities with mechanisms to ease the addressing of context changes and to facilitate the development of adaptive behavior. To overcome these limitations, we propose to empower BPEL with Aspect Oriented Programming (AOSD, 2007) to deal with Service Domain adaptation based on context. Our approach aims to prove the benefits of bringing Aspect Oriented paradigms to ensure context-aware services.

4 SD ADAPTABILITY USING ASPECTS

Our goal is to present an adaptable Service Domain to context changes. To this end, we use the Aspect Oriented Programming (AOP).

4.1 Motivation behind the AOP

Aspect Oriented Programming is an emerged paradigm which enables capturing and modularizing

concerns which crosscut a software system into modules called Aspects. Aspects can be integrated dynamically to the system thanks to dynamic weaving principle (AOSD, 2007). AOP introduces a unit of modularity called Aspects containing different code fragments (*advice*), and location descriptions (*pointcuts*) to identify where to plug the code fragment. These points, which can be selected by the pointcuts, are called *join points*.

The rationale behind using AOP is based on two arguments. First, AOP enable crosscutting concerns which is crucial for managing context information separately from the business logic implemented in the BPEL process. This separation of concerns makes the modification of context information and its related adaptability action easier. For example, in the Delivery Service Domain, we can define an Aspect related to the calculation of extra fees when there is a context change that corresponds to modifying the delivery date. This Aspect can be reused in several BPEL processes. Besides, we can attach the adaptability action (action realised as response to context change requirements) to another context information (eg. location context) without changing the orchestration logic.

Second, based on dynamic weaving principles, Aspects can be activated and deactivated at runtime. Consequently, BPEL process can be dynamically adapted at runtime.

Adding AOP to BPEL is very beneficial. However, AOP is currently used on low level language extension (Kiczales et al., 2001). In order to exploit AOP for SD adaptation, AOP techniques need to be improved to support: (i) runtime activation of Aspects in the BPEL process to enable dynamic adaptation according to context changes, and (ii) Aspects selection to enable customer-specific contextualization of the Service Domain.

4.2 Empowering the SD with Aspect

The core of our approach is a runtime Aspects weaving that can be injected on the existing SD BPEL process, to achieve adaptable execution based on context changes. Our key contribution consists of encapsulating context information and the corresponding adaptation actions in a set of Aspects.

A BPEL process is considered as a graph *G(V,E)* where *G* is a DAG (Directed Acyclic Graph). Each vertex $v_i \in V$ is a Web service (Web service operation). Each edge *(u, v)* represents a logical flow of messages from u to v. If there is an edge *(u, v)*, then it means that an output message produced by *u* is used to create an input message to *v*.

In this work, we use this definition of BPEL, but it is extended by adding specific constructs. We identify three types of vertex: (i) context aware vertex, (ii) non context aware vertex and (iii) context manager vertex. Theses vertexes correspond respectively to Context aware IT Services, Non-Context aware IT Service and Context Manager Services. Context manager vertexes detect context changes and usually precede the context aware vertexes.

A Context aware IT Service (CITS) is a service which may have several configurations exporting different behaviors according to the specific context. *CITS* = {<*ID-CITS*, *Ctx, Asp*>} where *ID-CITS* is the identifier of CITS, *Ctx* is the name of a context and *Asp* is the Aspect related to this context.

We define an Aspect as *Asp*=< *ID-Asp, Entry-condition, Advice, Join-points*>. Where *ID-Asp* is the identifier of the Aspect, *Entry-condition* represents the condition where the Aspect can be used, *Advice* addresses the adaptability actions related to specific context information (add, parameterize and remove IT service(s)) and *Join-points* describe the set of vertexes where possible adaptations may be required in the BPEL process.

Our adaptation approach is a three-step process:

1. Context detection consists of checking the runtime context information, in order to detect possible context changes. These tasks are performed by the Context Manager Service which is developed as a Web service in the BPEL process.

2. Aspect Activation is responsible for the plug-in and the removal of pre-defined Aspects into the BPEL process using the Aspect Activator Module. The Aspect Activator Module is conceived as an extension to the BPEL engine as was done in (Charfi and Mezini, 2007). When running a process instance, the Aspect Activator receives the context change information from the Context Manager Service. Then it chooses and activates the appropriate Aspect that matches the values of the changed contextual information.

3. Updating original BPEL Process by activating the right Aspect which is executed in the BPEL process to create a contextualized process.

4.3 Running Scenario

The running scenario is related to a manufacturer of plush toys enterprise which receives orders from its clients during the Christmas period. Once an order is received, the firm proceeds to supply the different components of plush toys. When supplied components are available, the manufacturer begins

assembly operations. Finally, the manufacturer selects a logistic provider to deliver these products by the target due date. In this scenario, the focus will be only on the delivery service.

Assume that an inter-enterprise collaboration is established between the manufacturer of plush toys (service consumer) and a logistic enterprise (service provider). The logistic provider delivers parcels from the plush toys manufacturer warehouse to a specific location. The delivery service starts by picking merchandise from the customer warehouse (see Figure 4 step (i)). If there is no past interaction between the two parties involved, the delivery service verifies the shipped merchandise. Once verified, *putting merchandise in parcels service* is invocated, which is followed by a *computing delivery price service*. Finally, the service transports the merchandise in the business opportunity location at the delivery due date. The delivery service is considered as a Service Domain orchestrating five IT services: *Picking merchandise*, *Verifying merchandise*, *Putting merchandise in parcels*, *Computing delivery price* and *Delivering merchandise*. Figure 4 depicts the BPEL process modelled as a graph of the delivery service and the adaptation actions according to the context changes (step ii and step iii).

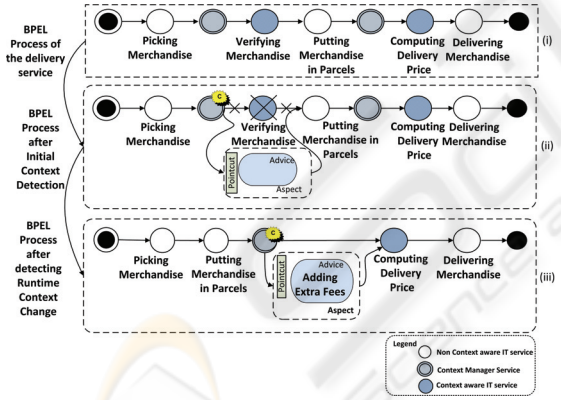

Figure 4: The delivery service internal process.

Assume that *Picking merchandise* from customer warehouse, *Putting merchandise in parcels* and *Delivering merchandise* services are context independent while *Verifying merchandise* and *Computing delivery price* are context-aware (i.e., they have different behaviors according to the current customer and opportunity context). We suppose that the *Verifying merchandise* service is aware of the past interactions with customers (historical relationships). This information corresponds to *history* category defined in the context categorization. This information may be either "past interaction=No" or "past interaction=Yes". In the first case, the *Verifying merchandise* service is called, but skipped in the second case, The *Computing delivery price* service is aware of runtime context changes corresponding to changes in delivery location or date. When there are changes in the date or the place, extra fees must be added to the total delivery price.

When the BPEL process (Listing 1) starts, the *Context Manager* service is invoked to collect context information (historical context category) about the plush toys enterprise (Listing 1 line 7). Assume that the context information indicates that the plush toys enterprise is a well known customer (i.e., "past interaction=Yes"). Delivery service behavior will be adapted to respond to this context information. The *Aspect Activator* will choose the suitable Aspect to be activated from the set of Aspects attached to the *Verifying merchandise* service.

```
(01) <process name = "DeliveryPackage" .../>
       <sequence>
(03)     <receive partner="client" operation="getDeliveryPackage"
           variable="request"  createInstance="yes" .../>
(05)     <invoke partner="PickingMerchandise" operation="getOkResponse"
           outputVariable="PickingResponse"/>
(07)     <invoke partner="ContextManager" operation="getContextElement"
           outputVariable="ContextResponse"/>
(09)     <invoke partner="VerifyingMerchandise" operation="VerifyMerchandises"
           outputVariable="VerifyingResponse />
(11)     <invoke partner="PuttingMerchandiseInParcels"
           operation="getOkResponse"  outputVariable="PuttingResponse"/>
(13)     <invoke partner="ContextManager" operation="getContextElement"
           outputVariable="ContextResponse"/>
(15)     <invoke partner="ComputingDeliveryPrice"
           operation="ComputePrice" outputVariable="PriceResponse"/>
(17)     <invoke partner="DeliveryMerchandise"
           operation="getOkResponse" outputVariable="DeliveryResponse"/>
(18)   </sequence>
         <assign>...</assign>
(19)     <reply partner="client" operation=" getDeliveryPackage " variable="proposition" .../>
     </sequence>
(21) </process>
```

Listing 1: The delivery process.

The selected Aspect is shown in Listing 2. As mentioned before, an Aspect defines one or more pointcuts and an advice. To implement the advice code, we have chosen the BPEL specification, as the goal is to adapt the BPEL process. For the pointcuts language, we have chosen XPath, a language specialized for addressing parts of an XML document (a BPEL process is an XML document). The advice part of the Aspect is expressed as a before advice activity, which is executed instead of the activity captured by the pointcut (line 3). The join point, where the advice is weaved, is the <invoke> activity that calls the Verifying merchandise service (line 5). The advice code is expressed as a <switch> activity. If *ContextResponse* ="1" (i.e., "past interaction=Yes") the advice branches to the activity <*empty*>, in order

to express that it is not really necessary to perform this service. The BPEL process of the *Delivery* service after applying the Aspect is depicted in the Figure 4 (step (ii)).

```
(01) <aspect name="If the enterprise is a partner, do not verify transported merchandise">
     <pointcutandadvice type= "before">
(03) <pointcut name="Verify Merchandise">
     //process[@name=" DeliveryPackage "]
(05) //invoke[@portType="VerifyPT"and
     @operation="VerifyMerchandise"]
(07) </pointcut>
     <advice>
(09)   <switch>
         <case condition="bpws:getVariableData
(11)     ('ThisProcess( ContextResponse )','PastInteraction') ==1">
           <empty/>
(13)     </case>
       </switch>
(15) </advice>
     </pointcutandadvice>
(17) </aspect>
```

Listing 2: Context as an Aspect.

Before invoking *Computing Delivery Price*, the *Context Manager* service checks the context information to detect possible context changes. Let us assume that the plush toys enterprise has decided to change the delivery date. Hence, the *Context Manager* service captures the new date (Listing 1 line13). Then the Aspect Activator chooses the suitable Aspect to be activated from the set of Aspects related to *Computing Delivery Price service*. The selected Aspect is shown in Listing 3. The pointcut of this Aspect (lines 3-6) selects the delivery price calculation activity in the delivery process. The context change is implemented using a *before* advice, which contains a switch with a case branch (lines 7-25) for calculating additional fees depending on the number of days between the initial and the new delivery date. This number will be multiplied by the daily fees already defined by the logistic enterprise. The case branch uses an assign activity (lines 14-21) to compute the additional fees to the part *ExtraFees* of the variable *calculPrice* which will be sent to the *Computing Delivery Price* service. The *Delivery* service BPEL process after applying the Aspect is depicted in the Figure 4 (step (iii)).

```
(01) <aspect name=" AddExtraFees ">
     <pointcutandadvice>
(03) <pointcut name="Fees calculation">
       //process[@name=" DeliveryPackage "]
(05)   //invoke[@operation="ComputePrice"]
     </pointcut>
(07) <advice type="before">
       <switch>
(09)     <case condition=
           "getVariableData ('( ContextResponse )',' NewDate') <
(11)       getVariableData ('( clientrequest )',' DeliveryDate')">
         <!-- Here comes the action implementation of the context change -->
(13)     <sequence name="Fees calculation">
         <assign>
(15)     <copy>
         <from expression="
(17)     ((getVariableData ('( ContextResponse )',' NewDate') -
         getVariableData ('( clientrequest )',' DeliveryDate ')) *100 >
(19)     <to variable="calculPrice" part="ExtraFees"/>
         </copy>
(21)     </assign>
         </sequence>
(23)   </case>
       </switch>
(25) </advice>
     </pointcutandadvice>
```

Listing 3: Managing context change as an Aspect.

5 RELATED WORK

There are many ongoing research efforts related to the adaptation of Web services and Web service composition according to context changes (Maamar et al., 2007; Bettini et al., 2007). In the proposed work, we focus specially on the adaptation of a BPEL (workflow) process. Some research efforts from the Workflow community address the need for adaptability. They focus on formal methods to make the workflow process able to adapt to changes in the environment conditions. For example, authors in (Casati et al., 2000) propose eFlow with several constructs to achieve adaptability. The authors use parallel execution of multiple equivalent services and the notion of generic service that can be replaced by a specific set of services at runtime. However, adaptability remains insufficient and vendor specific. Moreover, many adaptation triggers, like infrastructure changes, considered by workflow adaptation are not relevant for Web services because services hide all implementation details and only expose interfaces described in terms of types of exchanged messages and message exchange patterns. In addition, authors in (Modafferi et al., 2005) extend existing process modeling languages to add context sensitive regions (i.e., parts of the business process that may have different behaviors depending on context). They also introduce context change patterns as a mean to identify the contextual situations (and especially context change situations) that may have an impact

on the behavior of a business process. In addition, they propose a set of transformation rules that allow generating a BPEL based business process from a context sensitive business process. However, context change patterns which regulate the context changes are specific to their running example with no-emphasis on proposing more generic patterns.

There are a few works using an Aspect based adaptability in BPEL. In (Charfi and Mezini, 2007), the authors presented an Aspect oriented extension to BPEL: the AO4BPEL which allows dynamically adaptable BPEL orchestration. The authors combine business rules modeled as Aspects with a BPEL orchestration engine. When implementing rules, the choice of the pointcut depends only on the activities (invoke, reply or sequence). However in our approach the pointcut depends on the returned value of the *Context Manager* Web service which detects a context changes. Business rules in this work are very simple and do not express a pragmatic adaptability constraint like context change in our case. Another work is proposed in (Erradi et al., 2005) in which the authors propose a policy-driven adaptation and dynamic specification of Aspects to enable instance specific customization of the service composition. However, they do not mention how they can present the Aspect advices or how they will consider the pointcuts.

6 CONCLUSIONS

In this paper, we have presented a high-level structure called Service Domain which orchestrates a set a of related IT services based on BPEL specifications. The Service Domain envision enhances the Web service concept to satisfy the inter-enterprise collaboration requirements by hiding complexities of managing several fine grained IT services and also be presenting a context-aware behaviours. However, BPEL presents limits regarding to dynamic adaptation. To overcome this shortage, we use Aspect Oriented Programming principles especially crosscutting concerns and dynamic weaving. As future work, we are working towards completing the development of the Service Domain architecture. An empirical study to validate and test the proposed approach will be the driver of future research, and interaction with industrial partners is the key idea behind the validation of the proposed approach.

REFERENCES

Andrews, T. and Curbera, F., 2003. Business Process Execution Language for Web Services (BPEL4WS) version 1.1.

AOSD, 2007. Aspect–Oriented Software Development.

Bechhofer, S., Harmelen, F. I., 2004. OWL Web Ontology Language Reference.

Bettini, C., Maggiorini, D. and Riboni, D., 2007. Distributed Context Monitoring for the Adaptation of Continuous Services. *World Wide Web,* 10**,** 503-528.

Bradely, N. A. and Dunlop, M. D., 2005. Toward a Multidisciplinary Model of Context to Support Context-Aware Computing. *Human-Computer Interaction,* 20 (4), 403-446

Byrd, T. A. and Turner, D. E., 2001. An exploratory examination of the relationship between flexible IT infrastructure and competitive advantage. *Information and Management* 39 (1), 41-52.

Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V. and Shan, M.-C., 2000. Adaptive and Dynamic Service Composition in eFlow. *Proceedings of CAISE 2000.* Stockholm, Sweden.

Charfi, A. and Mezini, M., 2007. AO4BPEL: An Aspect-oriented Extension to BPEL. *World Wide Web,* 10**,** 309-344.

Dey, A. K., Abowd, G. D. and Salber, D., 2001. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction,* 16 (12), 97-166.

Erradi, A., Maheshwari, P. and Padmanabhuni, S., 2005. Towards a Policy-Driven Framework For Adaptive Web Services Composition. *Proceedings of the International Conference on Next Generation Web Services Practices* Seoul, Korea.

Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., 2001. Overview of AspectJ. *Proceedings of The European Conference on Object Oriented Programming* Budapest, Hungary.

Maamar, Z., Benslimane, D., Thiran, P., Ghedira, C., Dustdar, S. and Sattanathan, S., 2007. Towards a context-based multi-type policy approach for Web services composition. *Data & Knowledge Engineering,* 62 (2), 327-351.

Modafferi, S., Benatallah, B., Casati, F. and Pernici, B., 2005: A Methodology for Designing and Managing Context-Aware Workflows. *Proceedings of IFIP International conference MOBIS.*

Mostafaoui, S. and Mostafaoui, G. K., 2003. Towards A Contextualisation of Service Discovery and Composition for Pervasive Environments. *In Workshop on Web-services and Agent-based Engineering.*