An Aspect Oriented Approach for Context-Aware Service Domain Adapted to E-Business

Khouloud Boukadi[1], Chirine Ghedira[2], and Lucien Vincent[1]

[1] Division for Industrial Engineering and Computer Sciences, ENSM, Saint-Etienne, France
[2] LIRIS Laboratory, Claude Bernard Lyon 1 University, Lyon, France
{boukadi,Vincent}@emse.fr, cghedira@liris.cnrs.fr

Abstract. This paper proposes an architecture for a high-level structure called Service Domain which orchestrates a set a of related IT services based on BPEL specification. Service Domain was developed to enhance the Web service concept to suit e-business collaboration. Service Domains are developed to be context aware. Our approach highlights the benefits of bringing Aspect Oriented Programming to ensure context aware services. Thus, context awareness is guaranteed by enhancing BPEL execution using Aspect oriented paradigms. The proposed approach is illustrated with a running example that shows how Service Domain presents different behaviours according to the context changes.

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

1 Introduction

Today, enterprises are operating in a rapidly changing market characterized by increasing customer demand for low cost and short time to market. To cope with these business conditions, enterprises have adopted a two-level solution. The first alternative is found on the inter-organizational side, in which enterprises collaborate in e-business scenarios in order to provide the best products or services. Secondly, on the organizational side, enterprises must be more dynamic, flexible, and context-aware than ever to survive.

In this endeavor, enterprise information technology (IT) systems play a crucial role. The challenge for IT infrastructures has been to help companies to respond to changes that 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 corresponding flexible IT systems, which allow enterprises to collaborate with partners dynamically. Flexible IT systems are those that are malleable enough to deal with context changes in an unstable environment [1].

A contemporary approach for addressing these critical issues is the Service-Oriented Architecture and Web service technology, which offers a suitable technical foundation to ensure the flexibility required for IT systems [2]. 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 hereafter IT services.

1.1 Limitations of the Traditional IT Service Solution

IT services are published for internal or external use. They 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 consumers will undertake several 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 makes no sense for the service provider. To overcome these limitations, we believe that an enterprise must re-organize its IT services and presents its functionalities through a high-level service. In this work we develop a high level structure called *Service Domain* (SD), which logically represents a combination of related IT services as a single service. Service Domain orchestrates a set of IT services in order to provide 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 functions provided through the Web. Indeed, it must have the capacity to adapt its own behavior by comporting appropriately to accommodate the situation in which it evolves. To meet to this objective, Service Domain has to assess its current capabilities, ongoing commitments, and surrounding environment. As a result Service Domain must be context aware.

1.2 Contribution and Paper Organization

The contribution in this paper is twofold. In the First part, the architecture of a high-level structure named Service Domain is presented. It orchestrates a set a of related IT services based on the Business Process Execution Language (BPEL) specification [3]. In the second part, we address our context aware Service Domain. Context awareness is guaranteed by enhancing BPEL execution using Aspect Oriented Programming [4].

The rest of the paper is organized as follows. The Service Domain architecture is presented in Section 2. Then, in Section 3, we present our context categorization and highlight the drawbacks of BPEL in addressing adaptability to the context changes. In section 4, the Aspect Oriented Programming and how it is used to enhance BPEL adaptability are introduced. In addition, we present a running example and implementations. Section 5 details some related work. Finally, a conclusion and possible further work is proposed.

2 Service Domain Concept (SD)

The motivation behind the Service Domain concept is to achieve manageability when dealing with a large number of IT services. The Service Domain 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 mask 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).

In the future, our 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 Fig.1).

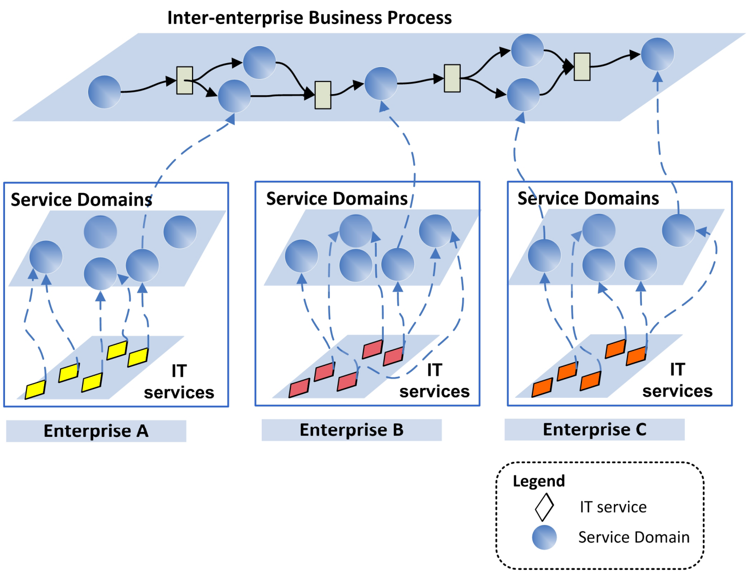

Fig. 1. Inter-enterprise collaboration based on Service Domain

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, consider the "logistic enterprise" that exposes a "Delivery Service Domain" (DSD), which constitutes a 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 services in one place facilitates manageability and avoids extra composition work on the client side as well as exposing non-significant services like "Verifying merchandise" on the enterprise side.

The Service Domain is implemented as a node consisting of an Entry Module, Context Manager Module (CMM), Service Orchestration Module (SOM) and finally an Aspect Activator Module (AAM) as presented in Fig. 2.

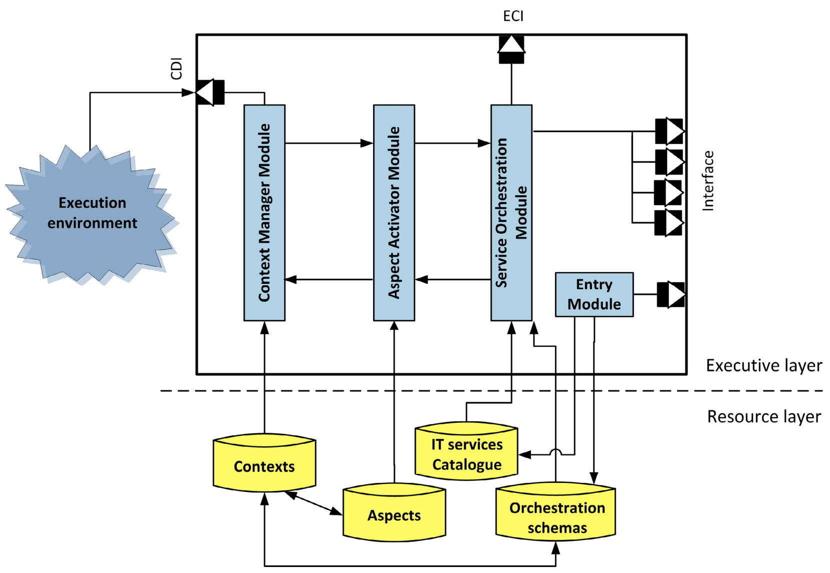

Fig. 2. Service Domain architecture

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 in the corresponding IT service catalogue. The register command can also deal with a new orchestration schema which could 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 Web service orchestration languages like BPEL [3]. 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. This is the 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 fulfill 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 Context and BPEL Adaptability

The trend towards context-aware, adaptive and on demand computing requires that SD be equipped 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/or unpredictable events).

SD must meet 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 Service Domain 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 present the context paradigm, our context categorization and finally, discuss the shortcomings of BPEL to address the adaptability to context requirement.

3.1 Context and Context Categorization

The concept of context has been studied in various fields for quite a long time. There are a number of different definitions and uses for this term. Context appears in many disciplines as a meta-information which characterizes the specific situation of an entity, to describe a group of conceptual entities, partition a knowledge base into manageable sets, or as a logical construct to facilitate reasoning services [5]. Our definition of context follows that of Dey's [6] 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 precisely by entities involved in the context. Bradely et al. depict that a variety of categorizations of context have also been proposed [7]. As a matter of fact, there are certain types of context which are, in practice, used more often than others. These major context categories are *location*, *identity*, *time,* and *activity*. Nevertheless, despite the various attempts to develop categorizations for context, there is no generic context categorization. Relevant information differs from one domain to another and depends on their effective use [8].

In this work, we propose a context categorization using an OWL ontology [9]. Fig. 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, customer, and collaboration related.

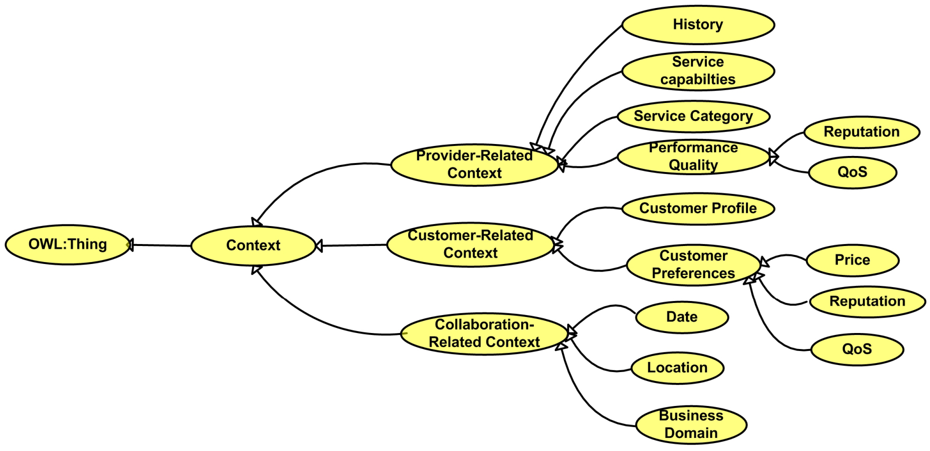

Fig.3. Ontology for categories of context

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

- Provider-related context deals with the conditions under which providers can offer their Web services externally. For example, the performance qualities include some metrics which measure the service quality: time, cost, QoS, and reputation. These attributes model the competitive advantages that providers may have over each other.
- Customer-related context represents the set of available information and meta-data used by service providers to adapt their services. For example, a customer profile which represents a set of information items characterizing the customer.
- 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 Adaptability to Context in BPEL

BPEL inherited a static view of the world from workflow management systems, which did not properly support evolutionary and runtime changes [10]. Only by stopping the running process, modifying the orchestration, and restarting process execution can one simulate evolutionary and runtime changes. Obviously, 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. Moreover, with BPEL 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. These limitations motivate 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 shortcomings, we propose to empower BPEL with Aspect Oriented Programming (AOP) [4] to deal with Service Domain adaptation based on context. Our approach shows the straightforwardness and benefits of bringing Aspect Oriented paradigms to ensure context aware services.

4 Service Domain Adaptability Using Aspects

The proposed approach defines and implements a context adaptive Service Domain using the Aspect oriented Programming (AOP) [4].

4.1 Rationale of AOP

AOP is a paradigm that captures and modularizes concerns that crosscut a software system into modules called Aspects. Aspects can be integrated dynamically to the system thanks to dynamic weaving principle [11].

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 most popular Aspect language is AspectJ [12] which is based on Java. The pointcut language of AspectJ provides a set of pointcut designators such as call (for selecting method), execution (for selecting method execution), get and set (for selecting read/write field access). However, each class of application can have its own specific Aspect implementation [13]. For instance, in aspect-oriented workflow languages, the advice language should be the same as the base workflow language [14] to avoid any paradigm mismatches for the workflow designers. There are some proposals to introduce some supplemental programming language in the BPELJ [15] [16], like adding Java code snippets to the BPEL engine.

The rationale behind using AOP is based on two arguments. First, AOP enables 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 the 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 executed as response to context change requirements) to different 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, the BPEL process can be dynamically altered at runtime.

Adding AOP to BPEL is very beneficial. However, AOP is currently used on a low level language extension [17]. In order to exploit AOP for SD adaptation, AOP techniques need to be improved to support:

- Runtime activation of Aspects in the BPEL process to enable dynamic adaptation according to context changes, and
- Aspects selection to enable customer-specific contextualization of the Service Domain.

4.2 Aspect Service Domain Specification

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. Three types of vertex were identified: (i) context aware, (ii) non context aware and (iii) context manager vertexes. 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) 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 (see Fig.4):

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 [18]. 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.

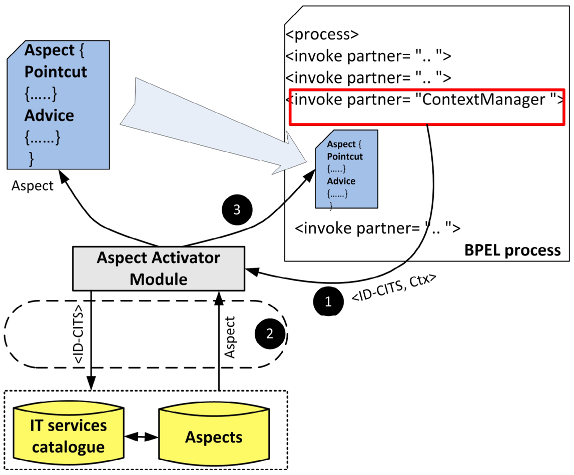

Fig.4. Aspect injection in BPEL

4.3 Running Example and Implementation

In this section, the proposed approach is applied to the case of a manufacturer of plush toys enterprise, which receives orders from its clients during the Christmas period. Once an order is received, this 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 Fig.5 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 invoked, and 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*,

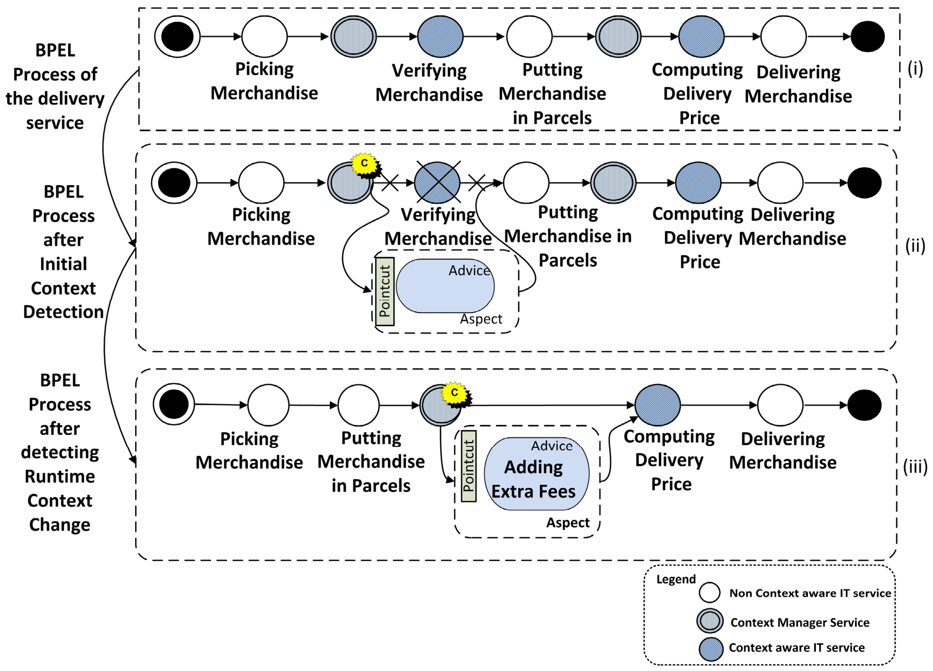
BPEL Process of the delivery service
Picking Merchandise
Verifying Merchandise
Putting Merchandise in Parcels
Computing Delivery Price
Delivering Merchandise
(i)
BPEL Process after Initial Context Detection
C
Pointcut
Advice
Aspect
(ii)
BPEL Process after detecting Runtime Context Change
Adding Extra Fees
(iii)
Legend
Non Context aware IT service
Context Manager Service
Context aware IT service

Fig. 5. The delivery service internal process

Computing delivery price and *Delivering merchandise*. Fig.5 depicts the BPEL process modeled as a graph of the delivery service and the adaptation actions according to the context changes (step ii and step iii).

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. It 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). Assuming 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 a 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, because the goal is to adapt the BPEL process. For the pointcuts language, we have chosen XPath [19], 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. After applying the Aspect, the BPEL process of the *Delivery* service is depicted in the Fig. 5 (step (ii)).

Before invoking *Computing Delivery Price*, the *Context Manager* service checks the context information to detect possible contextual 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 line 13). Then the *Aspect Activator* chooses a suitable Aspect to activate 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

```
(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

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 Fig. 5 (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 [20] [21]. In the proposed work we focus specially on the adaptation of a BPEL (workflow) process.

Some other 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, Casati et al. in [22] 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 considered by workflow adaptation, like infrastructure changes, 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, Modafferi et al. in [23] 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 behaviour of a business process. In addition, they propose a set of transformation rules that generate 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 [10, 24], 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 don’t express a pragmatic adaptability constraint like context change in our case. Another work is proposed in [25] ,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 don't mention how they can present the Aspect advices or how they will consider the pointcuts.

6 Conclusion

This paper presented an architecture of a high-level structure called Service Domain, which orchestrates a set of related IT services based on BPEL specification. Service Domain enhances the Web service concept to suit the inter-enterprise collaboration scenario. Besides, in order to address enterprise adaptability to context changes, Service Domain is developed to be context aware. Literature review has shown that BPEL, considered as the de facto standard for Web services orchestration, offers no support for dynamic adaptation of the orchestration logic according to context. To overcome these limitations, we proposed to enhance BPEL execution using the AOP. We demonstrated that it is a suitable paradigm that enables crosscutting and context-sensitive logic to be factored out of the service orchestration and modularized into Aspects. For future endeavors, we are working to improve, extend, and complete the Service Domain architecture. An empirical study to validate and test the proposed approach will be at the centre of future research. In addition, close interactions with industrial partners will be essential to validate the proposed approach.

References

1. Byrd, T.A., Turner, D.E.: An exploratory examination of the relationship between flexible IT infrastructure and competitive advantage. Information and Management 39, 41–52 (2001)
2. Papazoglou, M.P., van den Heuvel, W.-J.: Service-oriented design and development methodology. International Journal of Web Engineering and Technology (IJWET) 2(4), 412–442 (2006)
3. Andrews, T., Curbera, F.: Business Process Execution Language for Web Services (BPEL4WS) version 1.1 (2003), http://www-128.ibm.com/developerworks/library/specification/ws-bpel
4. Aspect–Oriented Software Development (2007), http://www.aosd.net
5. Benslimane, D., Arara, A., Falquet, G., Maamar, Z., Thiran, P., Gargouri, F.: Contextual Ontologies: Motivations, Challenges, and Solutions. In: Fourth Biennial International Conference on Advances in Information Systems, pp. 168–176. Springer (ED), Izmir (2006)
6. Dey, A.K., Abowd, G.D., Salber, D.: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Human-Computer Interaction 16, 97–166 (2001)
7. Bradely, N.A., Dunlop, M.D.: Toward a Multidisciplinary Model of Context to Support Context-Aware Computing. Human-Computer Interaction 20, 403–446 (2005)
8. Mostetefaoui, S.K., Mostetefaoui, G.K.: Towards A Contextualisation of Service Discovery and Composition for Pervasive Environments. In: Workshop on Web-services and Agent-based Engineering (2003)
9. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I.: OWL Web Ontology Language Reference (2004), http://www.w3.org/TR/2004/REC-owl-ref-20040210
10. Charfi, A., Mezini, M.: An Aspect-oriented Extension to BPEL, World Wide Web, pp. 309–344 (2007)
11. Bockisch, C., Haupt, M., Mezini, M., Ostermann, K.: Virtual Machine Support for Dynamic Join points. In: Proceedings of the 3rd International Conference on Aspect-Oriented Software Development - AOSD 2004, Lancaster, UK, pp. 83–92 (2004)
12. The AspectJ Team, The AspectJ Programming Guide, AspectJ 1.2 edition (2007), http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/aspectj-home/doc/progguide/index.html
13. Deursen, A.V., Klint, P., Visser, J.: Domain-Specific Languages: An Annotated Bibliography. ACM SIGPLAN Notices 35(6), 26–35 (2000)
14. Braem, M., Verlaenen, K., Joncheere, N., Vanderperren, W., Van Der Straeten, R., Truyen, E., Joosen, W., Jonckers, V.: Isolating Process-Level Concerns Using Padus. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 113–128. Springer, Heidelberg (2006)

15. Courbis, C., Finkelstein, A.: Towards Aspect Weaving Applications. In: Proceedings of the 27th International Conference on Software Engineering, pp. 66–77. ACM Press, New York (2005)
16. BEA and IBM, BPELJ: BPEL for Java, Joint White Paper (2004), http://www-128.ibm.com/developerworks/library/specification/ws-bpelj/
17. Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.G.: An Overview of AspectJ. In: Knudsen, J.L. (ed.) ECOOP 2001. LNCS, vol. 2072, Springer, Heidelberg (2001)
18. Charfi, A., Mezini, M.: Aspect-oriented web service composition with A04BPEL. In: The European Conference on web Service, pp. 168–182. Springer, Germany (2004)
19. Clark, J., DeRose, S.: XML Path Language (XPath) 1.0. W3C Recommendation, November 16 (1999), http://www.w3.org/TR/xpath
20. Maamar, Z., Benslimane, D., Thiran, P., Ghedira, C., Dustdar, S., Sattanathan, S.: Towards a context-based multi-type policy approach for Web services composition. Data & Knowledge Engineering, 327–335 (2007)
21. Bettini, C., Maggiorini, D., Riboni, D.: Distributed Context Monitoring for the Adaptation of Continuous Services. World Wide Web 10(4), 503–528 (2007)
22. Casati, F., Shan, M.-C.: Dynamic and adaptive composition of e-services. Information Systems 26(3), 143–163 (2001)
23. Modafferi, S., Benatallah, B., Casati, F., Pernici, B.: A Methodology for Designing and Managing Context-Aware Workflows. Mobile Information Systems II, 91–106 (2005)
24. Charfi, A., Mezini, M.: Hybrid web service composition: business processes meet business rules. In: The 2nd international conference on Service oriented computing, pp. 30–38. ACM Press, New York (2004)
25. Erradi, A., Maheshwari, P., Padmanabhuni, S.: Towards a Policy-Driven Framework For Adaptive Web Services Composition. In: The International Conference on Next Generation Web Services Practices (NWeSP 2005), Seoul, Korea, pp. 33–38 (2005)