Context-Aware Data and IT Services Collaboration in E-Business

Khouloud Boukadi[1], Chirine Ghedira[2], Zakaria Maamar[3], Djamal Benslimane[2], and Lucien Vincent[1]

[1] Ecole des Mines, Saint Etienne, France
[2] University Lyon 1, France
[3] Zayed University, Dubai, U.A.E.
{boukadi,Vincent}@emse.fr, {cghedira,dbenslim}@liris.cnrs.fr, zakaria.maamar@zu.ac.ae

Abstract. This paper discusses the use of services in the design and development of adaptable business processes, which should let organizations quickly react to changes in regulations and needs. Two types of services are adopted namely Data and Information Technology. A data service is primarily used to hide the complexity of accessing distributed and heterogeneous data sources, while an information technology service is primarily used to hide the complexity of running requests that cross organizational boundaries. The combination of both services takes place under the control of another service, which is denoted by service domain. A service domain orchestrates and manages data and information technology services in response to the events that arise and changes that occur. This happens because service domains are sensible to context. Policies and aspect-oriented programming principles support the exercise of packaging data and information technology services into service domains as well as making service domains adapt to business changes.

Keywords: service, service adaptation, context, aspect-oriented programming, policy.

1 Introduction

With the latest development of technologies for knowledge management on the one hand, and techniques for project management on the other hand, both coupled with the widespread use of the Internet, today's enterprises are now under the pressure of adjusting their know-how and enforcing their best practices. These enterprises have to be more focused on their core competencies and hence, have to seek the support of other peers through partnership to carry out their non-core competencies. The success of this partnership depends on how business processes are designed as these processes should be loosely coupled and capable to cross organizational boundaries. Since the inception of the Service-Oriented Architecture (SOA) paradigm along with its multiple implementation technologies such as Jini services and Web services, the focus of the industry community has been on providing tools that would allow seamless and flexible application integration within and across organizational boundaries. Indeed,

A. Hameurlain et al. (Eds.): Trans. on Large-Scale Data- & Knowl.-Cent. Syst. I, LNCS 5740, pp. 91–115, 2009.

SOA offers solutions to interoperability, adaptability, and scalability challenges that today's enterprises have to tackle. The objective, here, is to let enterprises collaborate by putting their core services together, which leads to the creation of new applications that should be responsive to changes in business requirements and regulations. Nevertheless, looking at enterprise applications from a narrowed perspective, which consists of services and processes only, has somehow overlooked the data that these applications use in terms of input and output. Data identification and integration are left at a later stage of the development cycle of these applications, which is not very convenient when these data are spread over different sources including relational databases, silos of data-centric homegrown or packaged applications, XML files, just to cite a few [1]. As a result, data identification and integration turn out tedious for SOA application developers: bits and pieces of data need to be retrieved/updated from/in heterogeneous data sources with different interfaces and access methods. This situation has undermined SOA benefits and forced the SOA community to recognize the importance of adopting a data-oriented view of services. This has resulted into the emergence of the concept of data services.

In this paper, we look into ways of exposing IT applications and data sources as services in compliance with SOA principles. To this end, we treat a service as either an IT Service (ITS) or a Data Service (DS) and identify the necessary mechanisms that would let ITSs and DSs first, work hand-in-hand during the integration exercise of enterprise applications and second, engage in a controlled way in high-level functionalities to be referred to as Service Domains (SDs). Basically, a SD orchestrates ITSs and DSs in order to provide ready-to-use high-level functionalities to users. By ready-to-use we mean service publication, selection, and combination of fine-grained ITSs and DSs are already complete.

We define ITSs as active components that make changes in the environment using update operations that empower them, whereas DSs as passive component that return data only (consultation) and thus, do not impact the environment. In this paper we populate this environment with specific elements, which we denote by Business Objects (BOs). The dynamic nature of BOs (e.g., new BOs are made available, some cease to exist without prior notice, etc.) and the business processes that are built upon these BOs, requires that SDs should be flexible and sensible to all changes in an enterprise's requirements and regulations. We enrich the specification of SDs with contextual details such as execution status of each participating service (DS or ITS), type of failure along with the corrective actions, etc. This enrichment is illustrated with the de facto standard namely the Business Process Execution Language (BPEL) for service integration. BPEL specifies a business process behavior through automated process integration both within and between organizations. We complete the enrichment of BPEL with context in compliance with Aspect-Oriented Programming (AOP) principles in terms of aspect injection, activation, and execution and through a set of policies.

While context and policies are used separately in different SOA initiatives[2, 3], we examine in this paper their role in designing and developing SDs. This role is depicted using a multi-level architecture that supports the orchestration of DSs and ITSs in response to changes detected in the context. Three levels are identified namely executive, business, and resource. The role of policies and context in this architecture is as follows:

- The trend towards context-aware, adaptive, and on-demand computing requires that SDs should respond to changes in the environment. This could happen by letting SDs sense the environment and take actions.
- Policies manage and control the participation of DSs and ITSs in SDs to guarantee free-of-conflicts SDs. Conflicts could be related to non-sharable resources and semantic mismatches. Several types of policies will be required so that the particularities of DSs and ITSs are taken into account.

The rest of the paper is organized as follows. Section 2 defines some concepts and introduces a running example. Section 3 presents the multi-level architecture for service (ITSs and DSs) collaboration and outlines the specification of these services. Section 4 introduces the adaptation of services based on aspect as well as the role of policies first, in managing the participation of DSs and ITSs in SDs and second, in controlling the aspect injection within SDs. Prior to concluding in Section 6, related work is reported in Section 5.

2 Background

2.1 Definitions

IT Service. The term IT services is used very often nowadays, even though not always with the same meaning. Existing definitions range from the very generic and all-inclusive to the very specific and restrictive. Some authors in [4, 5] define IT services as a software application accessible to other applications over the Web. Another definition is provided by [6], which says that an IT service is provided by an IT system or the IT department (respectively an external IT service provider) to support business processes. The characteristics of IT services can vary significantly. They can comprise single software components as well as bundles of software components, infrastructure elements, and additional services. These additional services are usually information services, consulting services, training services, problem solving services, or modification services. They are provided by operational processes (*IT service processes*) within the IT department or the external service provider [7]. In this paper, we consider that IT services should be responsible for representing and implementing business processes in compliance with SOA principles. An IT service corresponds to a functional representation of a real-life business activity having a meaningful effect to end-users. Current practices suggest that IT services could be obtained by applying IT SOA methods such as SAP NetWeaver and IBM WebSphere. These methods bundle IT software and infrastructure and offer them as Web services with standardized and well-defined interfaces. We refer to Web services that result out of the application of such methods as enterprise IT Web services or simply IT services.

Data Service is, in a recent report from Forrester Research, "*an information service (i.e., data service) provides a simplified, integrated view of real-time, high-quality information about a specific business entity, such as a customer or product. It can be provided by middleware or packaged as an individual software component. The information that it provides comes from a diverse set of information resources, including*

operational systems, operational data stores, data warehouses, content repositories, collaboration stores, and even streaming sources in advanced cases'' [8]. Another definition suggests that a data service is "*a form of Web service, optimized for the real-time data integration demands of SOA. Data services virtualize data to decouple physical and logical locations and therefore avoid unnecessary data replication. Data services abstract complex data structures and syntax. Data services federate disparate data into useful composites. Data services also support data integration across both SOA and non-SOA applications*" [9]. Data services can be seen as a new class of services that sits between service-based applications and enterprises' data sources. By doing so, the complexity of accessing these sources is minimized, which lets application developers focus on the application logics of the solutions to develop. These data sources are the basis of different business objects such as customer, order, and invoice.

Business object is "*the representation of a thing active in the business domain, including at least its business name and definition, attributes, behavior, relationships, and constraints*" (OMG's Business Object Management Special Interest Group). Another definition by the Business Object Management Architecture (jeffsutherland.com/oopsla97/marshall.html) suggests that a business object could be defined through the concepts of purpose, process, resource, and organization. A purpose is about the rationale of a business. A process illustrates how this purpose is reached through a series of dependent activities. A resource is a computing platform upon which the activities of a process are executed. Finally, an organization manages resources in terms of maintenance, access rights, etc.

Policies are primarily used to express the actions to take in response to the occurrence of some events. According to [10], policies are "*information which can be used to modify the behavior of a system*". Another definition suggests that policies are "*external, dynamically modifiable rules and parameters that are input to a system so that this latter can then adjust to administrative decisions and changes in the execution environment*" [5]. In the Web services field, policies are treated as rules and constraints that specify and control the behavior of a Web service upon invocation or participation in composition. For example, a policy determines when a Web service can be invoked, what constraints are put on the inputs a Web service expects, how a Web service can be substituted by another in case of failure, etc. According to [11], policies could be at two levels. At the higher level, policies monitor the execution progress of a Web service. At the lower level, policies address issues like how Web services communicate and what information is needed to enable comprehensive data exchange.

Context *"… is not simply the state of a predefined environment with a fixed set of interaction resources. It is part of process of interacting with an ever-changing environment composed of reconfigurable, migratory, distributed, and multi-scale resources"[12].* In the field of Web services, context facilitates the development and deployment of context-aware Web services. Standard Web services descriptions are then, enriched with context details and new frameworks to support this enrichment are developed [13].

Aspect Oriented Programming. AOP is a paradigm that captures and modularizes concerns that crosscut a software system into modules called Aspects. Aspects can be

integrated dynamically into a system using the dynamic weaving principle [14]. In AOP, unit of modularity is introduced using aspects that contain different code fragments (known as *advice*) and location descriptions (known as *pointcuts*) that identify where to plug the code fragment. These points, which can be selected using pointcuts, are called *join points*. The most popular Aspect language is Java-based AspectJ [15].

2.2 Running Example/Motivating Scenario

Our running example is about a manufacturer of plush toys that gets extremely busy with orders during Christmas time. When an order is received, the first step consists of requesting from suppliers the different components that contribute to the production of the plush toys as per an agreed time frame. When the necessary components are received, the assembly operations begin. Finally, the manufacturer selects a logistic company to deliver these products by the due date. In this scenario, the focus is on the delivery service only.

Let us assume an inter-enterprise collaboration is established between the manufacturer (service consumer) and a logistic enterprise (service provider). This latter delivers parcels from the manufacturer's warehouse to a specific location (Fig. 1 - step (i)). If there are no previous interactions between these two bodies, the logistic enterprise verifies the shipped merchandise. Upon verification approval, *putting merchandise in parcels* service is immediately invoked. This one uses a data service known as *parcel service*, which checks the number of parcels to deliver. *Putting merchandise in parcels* service is followed by delivery price and computing delivery price data services. The delivery price data service retrieves the *delivery price* that corresponds to the manufacturer order based on the different enterprise business objects it has access to such as toy (e.g., size of toy), customer (e.g., discount for regular customers), and parcel (e.g., size of parcel).

Finally, the merchandise is transported to the specified location at the delivery due date. The delivery service is considered as a SD that orchestrates four IT services and two data services: *picking merchandise*, *verifying merchandise*, *putting merchandise in parcels*, *delivery price*, *computing delivery price*, and *delivering merchandise*. Fig.1 depicts a graph-based orchestration schema (for instance a BPEL process) of the delivery service.

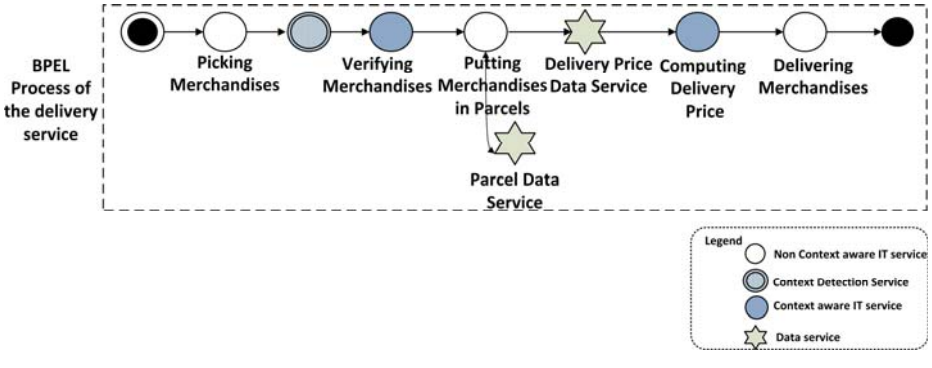

Fig. 1. The delivery service internal process

The inter-enterprise collaboration between the manufacturer and the logistic enterprise raises the importance of establishing dynamic (not pre-established) contacts since different enterprise logistics exist. To this end, the orchestration schema of the SD should be aware of the contexts of both manufacturer and nature of collaboration. Additional details can easily affect the progress of any type collaboration. For example, if the manufacturer is located outside the country, some security controls should be added and price calculation should be reviewed. Thus, the SD orchestration schema should be enhanced with contextual information that triggers changes in the process components (ITSs and DSs) in a timely manner. In this case, environmental context such as weather conditions (e.g., snow storm, heavy rain) may affect the IT service "*putting merchandise in parcels*". Consequently, several actions should be anticipated to avoid the deterioration of the merchandise by using metal instead of regular cardboard boxes.

Besides, the participation of the different ITSs and DSs in the delivery SD should be managed and controlled in order to guarantee that the obtained SD is free-of-conflicts. ITSs and DSs belong to different IT departments, each with its own characteristics, rules, and constraints. As a result, effective mechanisms that would ensure and regulate ITSs and DSs interaction are required. Besides, these mechanisms should also capture changes in context and guarantee the adaptation of service domains' behaviors in order to accommodate the situation in which they are going to operate.

3 Multi-level Architecture for Service Collaboration

In this section, the multi-level architecture that supports the orchestration of DSs and ITSs during the exercise of developing SDs is presented in terms of concepts, duties per layer, and service specifications.

3.1 Service Domain Concept

The rationale of the service domain concept is to abstract at a higher-level the integration of a large number of ITSs and DSs. A service domain is built upon (in fact, it uses existing standards such as WSDL, SOAP, and UDDI) and enhances the Web service concept. It does not define new application programming interfaces or standards, but hides the complexity of this exercise by facilitating service deployment and self-management. Fig. 2 illustrates the idea of developing inter-enterprise business processes using several service domains.

A SD involves ITSs and DSs in two types of orchestration schemas: vertical and horizontal. In the former, a SD controls a set of ITSs, which themselves controls a set of DSs. In the latter, a SD controls both ITSs and DSs at the same time. These two schemas offer the possibility of applying different types of control over the services whether data or IT. Businesses have to address different issues, so they should be given the opportunity to do so in different ways. DSs and ITSs participation in either type of orchestration is controlled through a set of policies, which permit to guarantee that SDs are free-of-conflicts. More details on policy use are given in subsections 4.2 and 4.3.2.

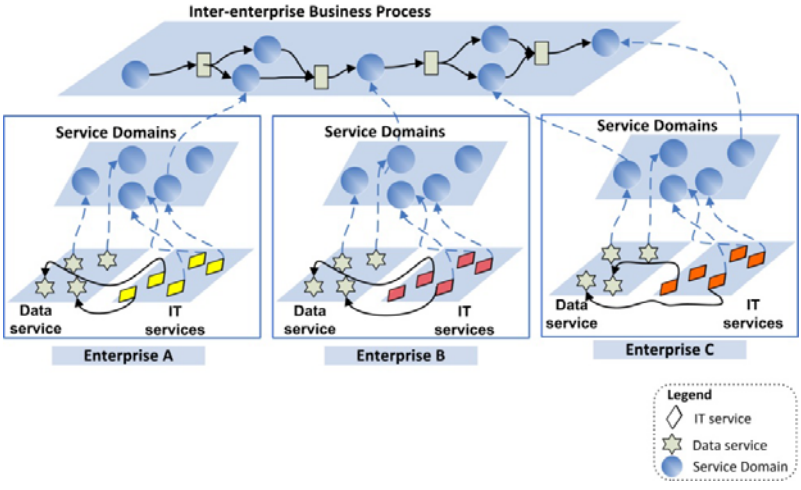

Fig. 2. Inter-enterprise collaboration based on service domains

According to the running example, the delivery SD orchestrates four IT services and two data services: *picking merchandise*, *verifying merchandise*, *putting merchandise in parcels*, *delivery price*, *computing delivery price*, and *delivering merchandise*. Keeping these ITSs and DSs 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.

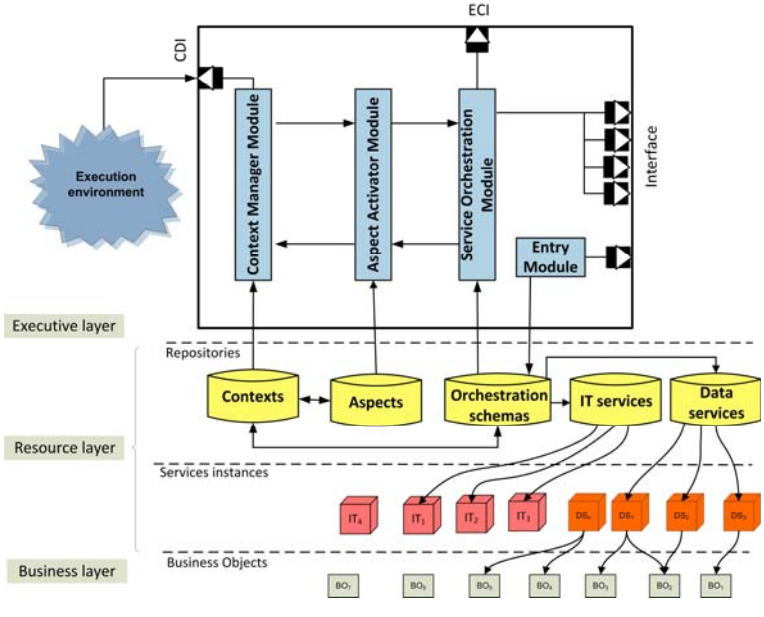

Fig. 3. Multi-layer architecture

The multi-level architecture in Fig.3 operates in a top-down way. It starts from executive-layer level, goes through the resource and business levels. These layers are described in detail in the following.

3.2 Roles and Duties per Layer

Executive layer. In this layer, a SD consists of an Entry Module (EM), a Context Manager Module (CMM), a Service Orchestration Module (SOM), and an Aspect Activator Module (AAM). In Fig. 3, CMM, SOM, and AAM provide external interfaces to the SD. The EM is SOAP-based to receive users' requests and return responses. In addition to these requests, the EM supports the administration of a SD. For example, an administrator can send a register command to add a new ITS to a given SD after signing up this ITS in the corresponding ITS registry. The register command can also be used to add a new orchestration schema to the orchestration schemas registry. When the EM receives a user's request, it screens the orchestration schemas registry to select a suitable orchestration schema for this request and identify the best ITSs and DSs. The selection of this schema and other services takes into account the customer context (detailed in section 4.1). Afterwards, the selected orchestration schema is delivered to the SOM, which is basically an orchestration engine based on BPEL [16].

The SOM presents an external interface called Execution Control Interface (ECI) that lets user obtain information about the status of a SD that is under execution. This interface is very useful in case of external collaboration as it ensures the monitoring of the SD progress. This is a major difference between a SD and a regular Web service. In fact, with the ECI a SD is based on the Glass box principle, which is opposite to the black box principle. In the glass box the SD way-of-doing is visible to the environment and mechanisms are provided to monitor the execution progress of the SD. Contrarily, a Web service is seen as a black box piece of functionality: described by its message interface and has no internal process structure that is visible to its environment.

Finally, the final external interface known as Context Detection Interface (CDI) is used by the CMM to detect and catch changes in context so that SD adaptability is guaranteed. This happens by selecting and injecting the right aspect with respect to the current context change. To this end, a SD uses the AAM to identify a suitable aspect for the current situation so that the AAM injects this aspect into the BPEL process.

Resource Layer. It consists of two sub-layers namely source and service instances.

- The sources sub-layer is populated with different registries namely orchestration schemas, ITS, DS, context, and aspect. The orchestration schemas registry consists of a set of abstract processes based on ITSs and DSs, which are at a later stage implemented as executable processes. In addition, this sub-layer includes a set of data sources that DSs use for functioning. The processing of these requests is subject to access privileges that could be set by different bodies in the enterprise such as security administrators. The content of the data sources evolves over time following data sources addition, withdrawal, or modification. According

to the running example, customer and inventory databases are examples of data sources.

- The services instances sub-layer is populated with a set of instance services that originate from ITSs and DSs. On the one hand, DS instances collect data from BOs and not from data sources. This permit to shield DSs from semantic issues and changes in data sources and to prepare data from disparate BOs that are scattered cross the enterprise. The data that a DS collects could have different recipients including SDs, ITSs, or other DSs. Basically, a DS crawls the business-objects level looking for the BOs it needs. A DS consults the states that the BOs are currently in to collect the data that are reported in these states. This collection depends on the access rights (either public or private) that are put on data; some data might not be available to DSs. According to the running example, a DS could be developed to track the status of the parcels included in the delivery process (i.e., number of used parcels). This DS would have to access order and update parcel BOs. If order BO takes now on *orderChecked* state, the DS will know the parcels that are confirmed for inclusion in the delivery and hence, interact with the right ITS to update the parcel BO. This is not the case if this BO was still in *orderUpdated* state; some items might not have been confirmed yet. On the other hand, ITS instances implement business processes that characterize enterprises' day-to-day activities. ITSs need the help of DSs and BOs for the data they produce and host, respectively. Update means, here, make a BO take on a new state, which is afterwards reflected on some data sources by updating their respective data. This is not the case with DSs that consult BOs only.

Business Layer. It (1) tracks the BOs that are developed and deployed according to the profile of the enterprise and (2) identifies the capabilities of each BO. In [17] we suggest that BOs should exhibit a goal-driven behavior instead of just responding to stimuli. The objective is to let BOs (i) screen the data sources that have the data they need, (ii) resolve data conflicts in case they raise, and (iii) inform other BOs about their capabilities of data source access and data mediation. As mentioned earlier, order, parcel, and customer are examples of BOs. These BOs are related to each other, e.g., an order is updated only upon inspection of customer record and order status. The data sources that these BOs access could be customer and inventory databases.

3.3 Layer Dependencies

Executive, resource, and business layers are connected to each other through a set of dependencies. We distinguish two types of dependencies: intra-layer dependencies and inter-layers dependencies.

1. **Intra-layer dependencies:** Within the resource layer two types of intra-layer dependencies are identified:

- **Type and instance dependency:** we differentiate between the ITSs types that are published in the IT services registry, which are groups of similar (in term of functionality) IT services, and the actual IT service instances that are made available for invocation. To illustrate the complexity of the dependencies that arises, we suggest hereafter a simple illustration of the number of IT service instances that could be obtained out of ITSs. Let us consider ITSt = {ITSt1, . . ., ITStα} a set of α IT service types and ITSi = {ITSi1, . . ., ITSiβ} a set of β service instances that exist in the services instances sub-layer. The mapping of S onto I is subjective and one-to-many. Assuming each IT service type ITst1 has N instantiations, $\beta = N \times \alpha$. The same dependency exists between DS types in the sources sub-Layer and data services instances in the services instances sub-layer.

    - **Composition dependency** involves orchestration schemas in two sub-layers namely source and service instance. Composition illustrates today's business processes that are generally developed using distributed and heterogeneous modules for instance services. For an enterprise business process, it is critical to identify the services that are required, the data that these services require, make sure that these services collaborate, and develop strategies in case conflicts occur.

2. **Inter-layers dependencies**

    - **Access dependency** involves the service instances sub-layer and the business layer. Current practices expose services directly to data sources, which could hinder the reuse opportunities of these services. The opposite is here adopted by making services interact with BOs for their needs of data. For a service, it is critical to identify the BOs it needs, comply with the access privileges of these BOs, and identify the next services that it will interact with after completing an orchestration process. Access dependency could be of different types namely on-demand, periodic, or event-driven. In the on-demand case, services submit requests to BOs when needed. In the periodic case, services submit requests to BOs according to a certain agreed-upon plan. Finally, in the event-driven case services submit requests to BOs in response to some event.
    - **Invocation dependency** involves the business and source layers. An invocation implements the technical mechanisms that allow a BO access the available data sources in terms of consultation or update (see Fig. 4). A given BO includes a set of operations:

        - A set of read methods, which provide various ways to retrieve and return one or more instances of the data included in a data source.
        - A set of write methods, responsible for updating (inserting, modifying, deleting) one or more instances of the data included in the data sources.
        - A set of navigation methods, responsible for traversing relationships from one data source to one or more data of a second data source. For example, the Customer BO can have two navigation methods *getDelivOrder* and *getElecOrder*, to fetch for a given customer the delivery orders from a delivery database and electronic orders from electronic order database.

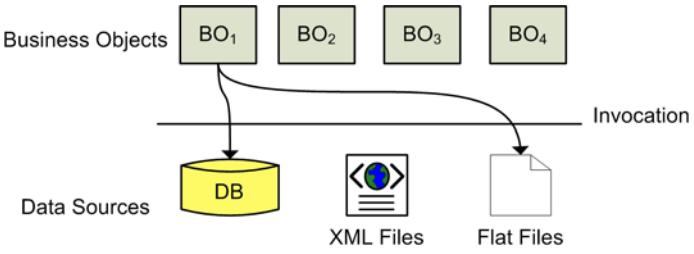

Fig. 4. Invocation dependency between the business objects and the data sources

3.4 Services Specifications

3.4.1 Data Service Specification

DSs come along with a good number of benefits that would smooth the development of enterprise SD:

Access unification: data related to a BO might be scattered across independent data sources that could present three kinds of heterogeneities:

- Model heterogeneity: each data source has its own data model or data format (relational tables, WSDL with XML schema, XML documents, flat files, etc.).
- Interface heterogeneity: each type of data source has its own programming interface; JDBC/SQL for relational databases, REST/SOAP for Web services, file I/O calls, and custom APIs for packaged or homegrown data-centric applications (like BAPI for SAP) .

The adoption of DSs relieves SOA application developers from having to directly cope with the first two forms of heterogeneity. That is, in the field of Web services all data sources are described using WSDL and invoked via REST or SOAP calls (which means having the same interface), and all data are in XML form and described using XML Schema (which means having the same data model).

Reuse and agility: the value-added of SOA to application development is reuse and agility, but without flexibility at the data tier, this value-added could quickly erode. Instead of relying on non-reusable proprietary codes to access and manipulate data in monolithic application silos, DSs can be used and reused in multiple business processes. This simplifies the development and maintenance of service-oriented applications and introduces easy-to-use capabilities to use information in dynamic and real-time processes.

To define DSs, we took into account the interactions that should take place between the DSs and BOs. DSs are given access to BOs for consultation purposes. Each DS represents a specific data-driven request whose satisfaction requires the participation of several BOs. The following suggests an example of itemStatusOrder DS whose role is to confirm the status of the items to include in a customer's order.

Table 1. DS service structure

```
DataService Class: itemStatusOrder {
Input:
      orderID              : string
      customerID           : string
Output:
      1[statusItem         : boolean]n
Methods:
      checkStatusItem      : (itemID)
}
```

In the above structure, the following arguments are used:

1. Input argument identifies the elements that need to be submitted to a DS. These elements could be obtained from different parties such as users and other DSs.
2. Output argument identifies the elements that a DS returns after its processing is complete.
3. Method argument identifies the actions that a DS implements in response to the access requests it runs over the different BOs.

Because DSs could request sensitive data from BOs, we suggest in Section 4.2.1 that appropriate policies should be developed so that data misuse cases are avoided. We refer to these policies as privacy.

3.4.2 IT Service Specification

For the IT service specification, we follow the one proposed by Papazoglou and Heuvel in [18] who specify an IT service as (1) a structural specification that defines service types, messages, port types, (2) a behavioral specification that defines service

Table 2. IT service structure

```
ITService Class: computing delivery price {
Input:
      orderID              : string
      customerID           : string
Output:
      Price                : Float
Methods:
      ComputePrice         : ()
Policies:
      Business policy      : link to a policy doc
      Behavior policy      : link to a policy doc
      Privacy policy       : link to a policy doc
      }
```

operations, effects, and side effects of service operations, and (3) a policy specification that defines the policy assertions and constraints on the service. Based on this specification, we propose in our work a set of policies as follows:

- Business policies correspond to policy specification.
- Behavior policy corresponds to structural specification.
- Privacy policies correspond to behavioral specification.

4 Services Collaboration

4.1 Context-Aware Orchestration

The concept of context appears in many disciplines as a meta-information that 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 [19]. The categorization of context is critical for the development of adaptable applications. Context includes implicit and explicit inputs. For example, user context can be deducted 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 that the context involves. Nevertheless, despite the various attempts to suggest a context categorization, there is no proper categorization. Relevant information differs from one domain to another and depends on their effective use [20]. In this paper, we propose an OWL-based context categorization in Fig. 4. This categorization is dynamic as new sub-categories can be added at any time. Each context definition belongs to a certain category, which can be related to provider, customer, and collaboration.

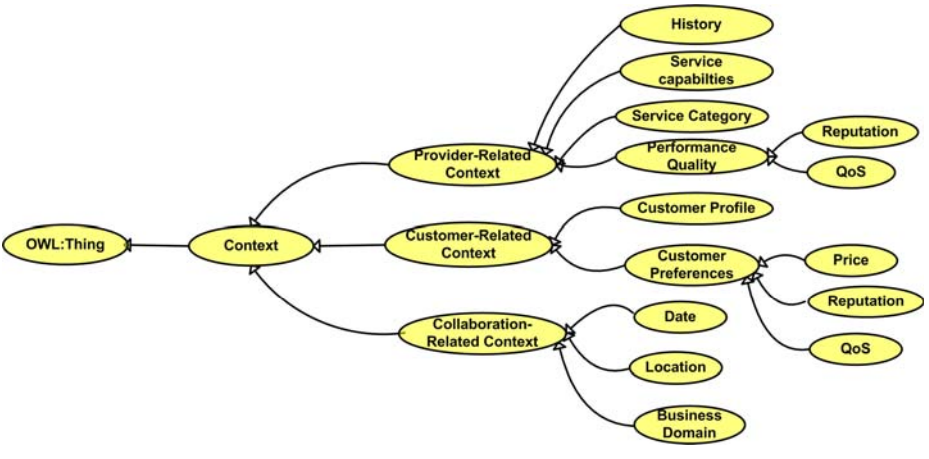

Fig. 5. Ontology for categories of context

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

- Provider-related context deals with the conditions under which providers can offer their SDs. For example, performance attributes including some metrics to measure a service quality: time, cost, QoS, and reputation. These attributes are used to model the competition between providers.
- Customer-related context represents the set of available information and metadata used by service providers to adapt their services. For example, a customer profile permits to characterize a user.
- 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.

4.2 Policy Specification

As stated earlier, policies are primarily used to first, govern the collaboration between ITSs, DSs, and SDs and second, reinforce specific aspects of this collaboration such as when an ITS accepts to take part in a SD and when a DS rejects a data request from an ITS because of risk of access right violation. Because of the variety of these aspects, we decompose policies into different types and dissociate policies from the business logics that services implement. Any change in a policy should "slightly'' affect a service's business logic and *vice versa*.

4.2.1 Types of Policies

Policies might be imposed by different types of initiators like the service itself, service provider, and user who plans to use the service [21].

- Service driven policy is defined by the individual organizational that offer services. This description is not enough.
- Service flow driven policy is defined by the organizations offering a composite web service.
- Customer driven policy is meant to future consumers of services. Generally, a user has various preferences in selecting a particular service, and these preferences have to be taken into account during composition or even during other steps like section selection, composition, and execution. For example, if two providers have two services with the same functionality, the user would like to consider the cheapest.

Policies are used in different application domains such as telecommunication, learning, just to cite a few, which supports the rationale of developing different types of policies. In this paper, we suggest the following types based on some of our previous works [11, 22]:

- **Business policy** defines the constraints that restrict the completion of a business process and determines how this process should be executed according to users' requirements and organizations' internal regulations. For example, a car loan

application needs to be treated within 48 hours and a bank account should maintain a minimum balance.

- **Behavior policy** supports the decisions that a service (ITS and DS) has to make when it receives a request from a DS to be part of the orchestration schema that this associated with this DS. In [22], we defined three behaviors namely permission, dispensation, and restriction, which we continue to use in this paper. Additional details on these behaviors are given later.
- **Privacy policy** safeguards against the cases of data misuse by different parties with focus here on DSs and ITSs that interact with BOs. For example, an ITS needs to have the necessary credentials to submit an update request to a BO. Credentials of an ITS could be based on the history of submitting similar request and reputation level.

Fig. 5 illustrates how the three behaviors of a service (DS or ITS) are related to each other based on the execution outcome of behavior policies [22]. In this figure, dispensation (P) and dispensation(R) stand for dispensation related to permission and related to restriction, respectively. In addition, engagement (+) and engagement (-) stand for positive and negative engagement in a SD, respectively.

- Permission: a service accepts to take part in a service domain upon validation of its current commitments in other service domains.
- Restriction: a service does not wish to take part in a service domain for various reasons such as inappropriate rewards or lack of computing resources.
- Dispensation means that a service breaks either a permission or a restriction of engagement in a service domain. In the former case, the service refuses to engage despite the positive permission that is granted. This could be due to the

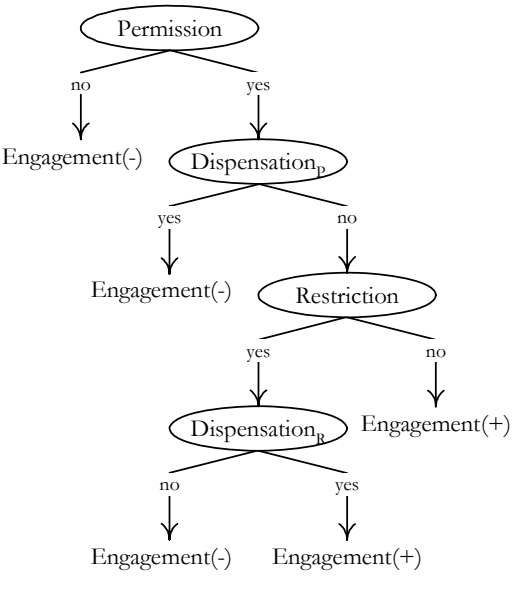

Fig. 6. Behaviors associated with a service

unexpected breakdown of a resource upon which the service performance was scheduled. In the latter case, the service does engage despite the restrictions that are detected. The restrictions are overridden because of the priority level of the business scenario that the service domain implements, which requires an immediate handling of this scenario.

In [11], Maamar et al. report that several types of policy specification languages exist. The selection of a policy specification language is guided by some requirements that need to be satisfied [23]: expressiveness to support the wide range of policy requirements arising in the system being managed, simplicity to ease the policy definition tasks for people with various levels of expertise, enforceability to ensure a mapping of policy specification into concrete policies for various platforms, scalability to guarantee adequate performance, and analyzability to allow reasoning about and over policies. In this paper we adopt WSPL is used. WSPL syntax is based on the OASIS eXtensible Access Control Markup Language (XACML) standard (www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0.pdf).

The Listing.1 suggests a specification of a behavior policy with focus on privacy in WSPL. It shows an example of an ITS that checks the minimum age and income of a person prior to approving a car loan application.

```
Policy (Aspect="Behavior "){
  <Rule xmlns="urn:oasis:names:tc:xacml:3.0:generalization:policy:schema:wd:01" RuleId="BusinessITS">
    <Condition>
      <Apply FunctionId="and">
        <Apply FunctionId="integer-less-than" DataType="boolean">
          <SubjectAttributeDesignator AttributeId="age" DataType="integer">
          <SubjectAttributeDesignator Attributeid="minimumAge" DataType="integer"/>
        </Apply>
        <Apply FunctionId="integer-less-than" DataType="boolean">
          <SubjectAttributeDesignator AttributeId="income" DataType="integer">
          <SubjectAttributeDesignator Attributeid="minimumIncome" DataType="integer"/>
        </Apply>
      </Apply>
    </Condition>
    <Conclusions>
      <TrueConclusion Approval = "Accept"/>
      <FalseConclusion  Approval = "Reject"/>
    </Conclusions>
  </Rule>}
```

Listing. 1. A behavior policy specification for an ITS

The Listing.2 suggests a specification of a business policy in WSPL. It shows an example of a DS that checks the possibility of taking part in a service domain.

In addition to the arguments that form WSPL-defined policies, we added additional arguments for the purpose of tracking the execution of these policies. These additional arguments are as follows:

- Purpose: describes the rationale of developing a policy P.
- Monitoring authority: identifies the party that checks the applicability of a policy P so that the outcomes of this policy are reinforced. A service provider or policy developer could illustrate these parties.

```
Policy (Aspect="Business"){
  <Rule xmlns="urn:oasis:names:tc:xacml:3.0:generalization:policy:schema:wd:01" RuleId="PermissionDS">
    <Condition>
      <Apply FunctionId="integer-less-than" DataType="boolean">
        <SubjectAttributeDesignator AttributeId="currentNumberofParticipations" DataType="integer">
        <SubjectAttributeDesignator Attributeid="maximumNumberofParticipations " DataType="integer"/>
      </Apply>
    </Condition>
    <Conclusions>
      <TrueConclusion Permission = "Permit"/>
      <FalseConclusion  Permission = "Deny"/>
    </Conclusions>
</Rule>}
```

Listing. 2. A business policy specification for a DS

– Scope (local or global): identifies the parties that are involved in the execution of a policy P. "Local" means that the policy involves a specific services, where "global" means that the policy involves different services.

– Side-effect: describes the policies that could be triggered following the completion of policy P.

– Restriction: limits the applicability of a policy P according to different factors such as time (e.g., business hours) and location (e.g., departments affected by policy P performance).

4.3 Service Domain Adaptability Using Aspects

In the following we describe how we define and implement a context adaptive service domain using AOP.

4.3.1 Rationale of AOP

AOP is based on two arguments. First, AOP enables crosscutting concerns, which is crucial to separate context information from the business logic. For example, in Delivery Service Domain, an aspect related to the calculation of extra fees could be defined in case there is a change in the delivery date. Second, AOP promotes the dynamic weaving principle. Aspects are activated and deactivated at runtime. Consequently, a BPEL process can be dynamically altered upon request.

For the needs of SD adaptation, we suggest the following improvements in the existing AOP techniques: 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.

4.3.2 Using Policies to Express Contexts

Modeling context is a crucial issue that needs to be addressed to assist context-aware applications. By context modeling we mean the language that will be used to define both service and enterprise collaboration contexts. Since, there is a diversity of contextual information, we find several context modeling languages such as *ConteXtML* [24], *contextual schemas* [25], *CxBR* (context-based reasoning) [26], and *CxG* (contextual graphs) [27]. These languages provide the means for defining context in specific application domains such as pervasive and mobile computing. All these representations have

strengths and weaknesses. As stated in [28], lack of generality is the most frequent drawback: usually, each representation is suited for only a specific type of application and expresses a narrow vision of the context. Consequently, they present little or no support for defining context in Web service based collaboration scenarios. In this paper, we model the different types of context based on policy. Relation between context and policies is depicted in the definitions below:

Definition 1. A service Context *Ctxt* is a pair (Ctxt-name, P) where Ctxt-name corresponds to the context name derived from the context ontology (Fig) and P is the policy related to the given context Ctxt. Let P-set= $\{P_1, P_2, \ldots, P_n\}$ denotes the set of policies and SCx= $\{Cx_1, Cx_2, \ldots, Cx_n\}$ the set of context properties related to a particular ITS or DS. We express the mapping between ITS or data service' contexts and policies with the mapping function MF_s: SCx→P-set which gives the policies related to a given ITS or Data service.

Definition 2. A customer context *Custxt* is a pair (*Custxt* -name, P) where Custxt-name corresponds to the context name derived from the context ontology (Fig) and P is the policy related to the given context Custxt. Let P-set= $\{P_1, P_2, \ldots, P_n\}$ denotes the set of policies and CCx= $\{Cx_1, Cx_2, \ldots, Cx_n\}$ the set of context properties related to a particular customer. Same as the definition 1, we define a mapping function which retrieves the set of policies relating to a given customer context: MF_c: CCx →P-set.

In these definitions, context is described with policies. Consequently, to express context we need to express at first policies. We introduce the specification of context (customer, collaboration, and service contexts) in WSPL. Introducing the context concept in WSPL comes from the need to specify certain constraints that can depend on the environment in which the customer, the service, and the business collaboration are operational. For instance, a customer context that depicts a security requirement can be specified as follows.

```
Policy (Aspect="Security") {
        Rule (Context = "Customer_Related_Context")
                {
                Customer_Security_preferences="Secured
transaction"
                }
}
```

Listing. 3. A customer context specified as a policy

4.3.3 Controlled Aspect Injection through Policies

We show how policies and AOP can work hand-in-hand Policies related to customer and collaboration contexts are used to control the aspect injection within a SD. A SD provides a set of adaptation actions that are context dependent. We implement these actions as a set of aspects in order not to create any invasive code in the functional service implementation. An aspect includes a pointcut that matches a given ITS or

data service and one or more advices. These advices refer to the context dependent adaptation actions of this service. Advices are defined as Java methods and pointcuts are specified in XML format.

Our implementation approach for the controlled aspect injection through policies is presented in Fig.7. In this figure, the Aspect Activator Module previously presented in Fig.3, includes the Aspect Manager Module (AMM), the Matching Module (MM), and the Weaver Module (WM).

- The AMM is responsible for adding new aspects to a corresponding aspect registry. In addition, the AMM can deal with a new advice implementation, which could be added to this registry. The aspect registry contains the method names of the different advices related to a given ITS or data service.
- The MM is the cornerstone of the proposed aspect injection approach. It receives matching requests from the AMM and returns one or a list of matched aspects.
- The WM is based on an AOP mechanism known as weaving. The WM performs a run time weaving, which consists of injecting an advice implementation into the core logic of an ITS or data service.

The control of an aspect injection into a DS or ITS is as follows. Once a context dependent IT service or Data service operation is reached, the Context Manager Module sends the AAM the service's ID and its context dependent operation' ID. Then, the AMM identifies the set of aspects that can be executed to the ITS or DS based on the information sent by the Context Manager Module (i.e., service's ID and Operation's ID) (action 1 and 2). The set of aspects as well as the customer policies are

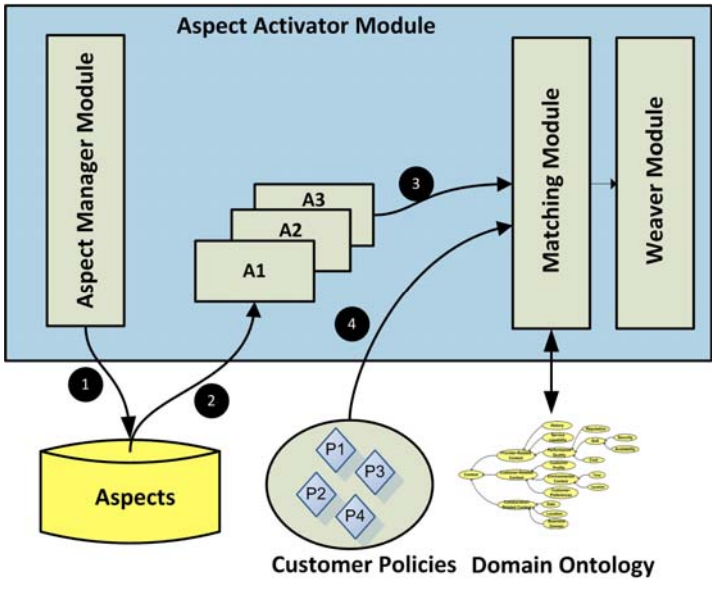

Fig. 7. Controlled aspect injection through policies

transmitted to Matching Module which returns the aspects that match the customer and the collaborations policies. The matching module is based on a matching algorithm and uses domain ontology. Finally, the WM integrates the advice implementation into the core logic of the service. By doing so, the service will execute the appropriate aspect in response to the current context (customer and collaboration contexts).

For illustration purposes, consider a payment ITS which is aware of the past interactions with customers. For loyal customer, the credit card payment is accepted, but bank transfer is required for new customers. Hence, the payment operation depends on the customer context, i.e., loyal or new one. The context dependent behaviors of the payment ITS are exposed as a set of aspects. Three of them are depicted in Listing.4.

```
(01) package aop. jboss ;
(02) package org. jboss .aop. advice ;
(03) import org. jboss. aop. advice . Interceptor;
(04) import org. jboss. aop. joinpoint. Invocation;
(05) import org. jboss. aop. joinpoint. MethodInvocation;
(06)  // aspect class declaration: Aspect for Payment ITService
(07)   Public class Aspect Aspect4Payment ITService {
(08)       @InterceptorDef (scope = Scope.PER_VM)
(09)       @Bind (pointcut="execution("* DDK.Test->Payment(..)" or "Customer= Loyal" )

(10)    Public Object PaymentOperation( Invocation invocation) throws Throwable {
(11)       Public String CreditCardPayment(RSA-SHL (Card-Num), OrderSum) {
(12)      return Confirmation;
(13)         }
(14)          return invocation. invokeNext();
(15)            }
(16)            }
```

///**Aspect 1**

```
(01) package aop. jboss ;
(02) package org. jboss .aop. advice ;
(03) import org. jboss. aop. advice . Interceptor;
(04) import org. jboss. aop. joinpoint. Invocation;
(05) import org. jboss. aop. joinpoint. MethodInvocation;
(06)  // aspect class declaration: Aspect for Payment ITService, Bank Transfer
(07)   Public class Aspect Aspect4Payment ITService {
(08)       @InterceptorDef (scope = Scope.PER_VM)
(09)       @Bind (pointcut="execution("* DDK.Test->Payment(..)" or "Customer= New" )

(10)    Public Object PaymentOperation( Invocation invocation) throws Throwable {
(11)       Public String bank transfer (BqNum, OrderSum) {
(12)      return Confirmation;
(13)         }
(14)          return invocation. invokeNext();
(15)            }
(16)            }
```

///**Aspect 2**

```
(01) package aop. jboss ;
(02) package org. jboss .aop. advice ;
(03) import org. jboss. aop. advice . Interceptor;
(04) import org. jboss. aop. joinpoint. Invocation;
(05) import org. jboss. aop. joinpoint. MethodInvocation;
(06)  // aspect class declaration: Aspect for Payment ITService -secured transaction
(07)   Public class Aspect Aspect4Payment ITService {
(08)       @InterceptorDef (scope = Scope.PER_VM)
(09)       @Bind (pointcut="execution("* DDK.Test->Payment(..)" or "Customer= Loyal" )

(10)    Public Object PaymentOperation( Invocation invocation) throws Throwable {
(11)       Public String CreditCardPayment(RSA-SHL (Card-Num), OrderSum) {
(12)      return Confirmation;
(13)         }
(14)          return invocation. invokeNext();
(15)            }
(16)            }
///Aspect 3
```

Listing. 4. The three aspects related to the payment ITS

For example, the advice of Aspect 1 is expressed as a Java class, which is executed instead of the operation captured by the pointcut (line 9). The join point, where the advice is weaved, is the payment operation (line 10). The pointcuts are expressed as a condition If *Customer* ="Loyal" (i.e., "past interaction=Yes") the advice uses the credit card number, in order to perform the customer payment. Consider now the customer related context which specifies a security requirement, which is previously described. Based on this requirement, when executing the payment service, the matching module will determine that only aspect 3 with secured transaction should be applied. This aspect is then transmitted to the weaver module in order to be injected in the payment service.

5 Related Work

In this work, we identify two types of works related to our proposal: on the one hand, those proposals that, come from the data engineering field and propose approaches for data service modeling and development; and, on the other hand, those ones that focus specially in the adaptation of ITS (Web services).

Data services & the SOA software industry. Data services have gained considerable attention from SOA software industry leaders over the last three years. Many products are currently offered or being developed to make the creation of Data services easier than ever, to cite a few, AquaLogic by BEA Systems [29], Astoria by Microsoft [30], MetaMatrix by RedHat [31], Composite Software [9], Xcalia [32], and IBM [33]. The products offered here integrate the enterprise's data sources and provide a uniform access to data through Data services. As a representative example,

AquaLogic BEA's data service is a collection of functions that all have a common output schema, accept different sets of parameters, and are implemented via individual XQuery expressions. In a simplified example, a Data Service exports a set of functions returning Customer objects where one function takes as input the customer's last name, another one her city and state, and so on. AquaLogic exports these Data services to SOA application developers as Data Web Services, where functions become operations.

In Microsoft's Astoria project a data service or ADO.NET data services is a REST-based framework that allows releasing data via flexible data services and well-known industry standards (JSON and Atom). As opposed to message-oriented frameworks like SOAP-based services, REST-based services use basic HTTP requests (GET, POST, PUT and DELETE) to perform CRUD standing for Create, Read, Update and Delete operations. Such query patterns allow navigating through data, following the links established with the data schema. For example, /Customers ('PKEY1')/Orders (1)/Employees returns the employees that created sales order 1 for the customer with a key of 'PKEY1. (Source: http://msdn.microsoft.com/en-us/library/cc907912.aspx)

In addition, most commercial databases products incorporate mechanisms to export database functionalities as Data Web services. Representative examples are the IBM Document Access Definition Extension (DADX) technology (Db2XMLextender[1]) and the Native XML Web Services for Microsoft SQL Server 2005[34]. DADX is part of the IBM DB2 XML Extender, an XML/relational mapping layer, and facilitates the development of Web services on top of relational databases that can, among other things, execute SQL queries and retrieve relational data as XML.

Web services adaptation. Regarding the adaptation of Web services according to context changes [35]; [36] many ongoing research have been released. In the proposed work, we focus specially on the adaptation of a 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 [37] 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 [38] 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.

[1] Go online to http://www.306.ibm.com/software/data/db2/extenders/xmlext/

There are a few works using an Aspect based adaptability in BPEL. In [39], 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). 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 [40] 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 Conclusion

In this paper, was presented a multi-level architecture that supports the design and development of a high-level type of service known as Service Domain. This one orchestrates a set of related ITSs and DSs. Service Domain enhances the Web service concept to tackle the challenges that E-Business collaboration poses. In addition, to address enterprise adaptability to context changes, we made Service Domain sensible to context. We enhanced BPEL execution with AOP mechanisms. We have shown that AOP enables crosscutting and context-sensitive logic to be factored out of the service orchestration and modularized into Aspects. Last but not least, we illustrated the role of policies and context in a Service Domain. Different types of policies were proposed and then used first, to manage the participation of DSs and ITSs in SDs and second, to control aspect injection within the SD. In term of future work, we plan to complete the SD multi-level architecture and conduct a complete empirical study of our approach.

References

1. Carey, M., et al.: Integrating enterprise information on demand with xQuery. XML Journal 2(6/7) (2003)
2. Yang, S.J.H., et al.: A new approach for context aware SOA. In: Proc. e-Technology, e-Commerce and e-Service, EEE 2005, pp. 438–443 (2005)
3. Gorton, S., et al.: StPowla: SOA, Policies and Workflows. In: Book StPowla: SOA, Policies and Workflows. Series StPowla: SOA, Policies and Workflows, pp. 351–362 (2007)
4. Arsanjani, A.: Service-oriented modeling and architecture (2004), http://www.ibm.com/developerworks/library/ws-soa-design1/
5. Erl, T.: Service-Oriented Architecture (SOA): Concepts, Technology, and Design, p. 792. Prentice Hall, Englewood Cliffs (2005)
6. Huang, Y., et al.: A Service Management Framework for Service-Oriented Enterprises. In: Proceedings of the IEEE International Conference on E-Commerce Technology (2004)
7. Braun, C., Winter, R.: Integration of IT Service Management into Enterprise Architecture. In: Proc. The 22th Annual ACM Symposium on Applied Computing, SAC 2007 (2007)

8. Gilpin, M., Yuhanna, N.: Information-As-A-Service: What's Behind This Hot New Trend? (2007), http://www.forrester.com/Research/Document/Excerpt/0,7211,41913,00.html
9. C. Software, SOA Data Services Solutions, technical report (2008), http://compositesoftware.com/solutions/soa.html
10. Lupu, E., Sloman, M.: Conflicts in Policy-Based Distributed Systems Management. IEEE Transactions on Software Engineering 25(6) (1999)
11. Zakaria, M., et al.: Using policies to manage composite Web services. IT Professional 8(5) (2006)
12. Coutaz, J., et al.: Context is key. Communications of the ACM 48(3) (2005)
13. Keidl, M., Kemper, A.: A Framework for Context-Aware Adaptable Web Services (Demonstration). In: Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K., Ferrari, E. (eds.) EDBT 2004. LNCS, vol. 2992, pp. 826–829. Springer, Heidelberg (2004)
14. AOP, Aspect-Oriented Software Development (2007), http://www.aosd.net
15. AspectJ, The AspectJ Programming Guide (2007), http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~aspectj-home/doc/progguide/index.html
16. Andrews, T., et al.: Business Process Execution Language for Web Services (2003), http://www.ibm.com/developerworks/library/specification/ws-bpel/
17. Maamar, Z., Sutherland, J.: Toward Intelligent Business Objects. Communications of the ACM 43(10)
18. Papazoglou, M.P., Heuvel, W.-J.v.d.: Service-oriented design and development methodology. International Journal of Web Engineering and Technology (IJWET) 2(4), 412–442 (2006)
19. Benslimane, D., Arara, A., Falquet, G., Maamar, Z., Thiran, P., Gargouri, F.: Contextual Ontologies: Motivations, Challenges, and Solutions. In: Yakhno, T., Neuhold, E.J. (eds.) ADVIS 2006. LNCS, vol. 4243, pp. 168–176. Springer, Heidelberg (2006)
20. Mostefaoui, S.K., Mostefaoui, G.K.: Towards A Contextualisation of Service Discovery and Composition for Pervasive Environments. In: Proc. the Workshop on Web-services and Agent-based Engineering (2003)
21. Dan, A.: Use of WS-Agreement in Job Submission (September 2004)
22. Maamar, Z., et al.: Towards a context-based multi-type policy approach for Web services composition. Data & Knowledge Engineering 62(2) (2007)
23. Damianou, N., Dulay, N., Lupu, E.C., Sloman, M.: The ponder policy specification language. In: Sloman, M., Lobo, J., Lupu, E.C. (eds.) POLICY 2001. LNCS, vol. 1995, pp. 18–38. Springer, Heidelberg (2001)
24. Ryan, N.: ConteXtML: Exchanging contextual information between a Mobile Client and the FieldNote Server, http://www.cs.kent.ac.uk/projects/mobicomp/fnc/ConteXtML.html
25. Turner, R.M.: Context-mediated behavior for intelligent agents. Human-Computer studies 48(3), 307–330 (1998)
26. Gonzales, A.J., Ahlers, R.: Context-based representation of intelligent behavior in training simulations. International Transactions of the Society for Computer Simulation, 153–166 (1999)

27. Brezillon, P.: Context-based modeling of operators' Practices by Contextual Graphs. In: Proc. 14th Mini Euro Conference in Human Centered Processes (2003)
28. Bucur, O., et al.: What Is Context and How Can an Agent Learn to Find and Use it When Making Decisions? In: Proc. international workshop of central and eastern europe on multi agent systems, pp. 112–121 (2005)
29. Carey, M.: Data delivery in a service-oriented world: the BEA aquaLogic data services platform. In: Proc. The 2006 ACM SIGMOD international conference on Management of data (2006)
30. C. Microsoft, ADO.NET Data Services (also known as Project Astoria) (2007), http://astoria.mslivelabs.com/
31. Hat, R.: MetaMatrix Enterprise Data Services Platform (2007), http://www.redhat.com/jboss/platforms/dataservices/
32. X. Inc, Xcalia Data Access Services (2009), http://www.xcalia.com/products/xcalia-xdas-data-access-service-SDO-DAS-data-integration-through-web-services.jsp
33. Williams, K., Daniel, B.: SOA Web Services - Data Access Service. Java Developer's Journal (2006)
34. Microsoft, Native XML Web services for Microsoft SQL server (2005), http://msdn2.microsoft.com/en-us/library/ms345123.aspx
35. Maamar, Z., et al.: Towards a context-based multi-type policy approach for Web services composition. Data & Knowledge Engineering 62(2), 327–351 (2007)
36. Bettini, C., et al.: Distributed Context Monitoring for the Adaptation of Continuous Services. World Wide Web 10(4), 503–528 (2007)
37. Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V., Shan, M.-C.: Adaptive and Dynamic Service Composition in eFlow. In: Wangler, B., Bergman, L.D. (eds.) CAiSE 2000. LNCS, vol. 1789, p. 13. Springer, Heidelberg (2000)
38. Modafferi, S., et al.: A Methodology for Designing and Managing Context-Aware Workflows. In: Mobile Information Systems II, pp. 91–106 (2005)
39. Charfi, A., Mezini, M.: AO4BPEL: An Aspect-oriented Extension to BPEL. World Wide Web 10(3), 309–344 (2007)
40. Erradi, A., et al.: Towoards a Policy-Driven Framework For Adaptive Web Services Composition. In: Proceedings of the International Conference on Next Generation Web Services Practices 2005, pp. 33–38 (2005)