A Synergy Between Context-aware Policies and AOP to Achieve Highly Adaptable Web Services

Hamdi Yahyaoui[1], Azzam Mourad[2], Mohamed Almulla[1],
Lina Yao[3], and Quan Z. Sheng[3]

[1]Computer Science Department
Kuwait University, Kuwait, State of Kuwait
{hamdi, almulla}@sci.kuniv.edu.kw

[2]Department of Computer Science and Mathematics
Lebanese American University, Beirut, Lebanon
azzam.mourad@lau.edu.lb

[3]School of Computer Science
The University of Adelaide, Adelaide SA 5005, Australia
{lina, qsheng}@cs.adelaide.edu.au

Abstract. Modern service-based systems are frequently required to be highly adaptable in order to cope with rapid changes and evolution of business goals, requirements, as well as physical context in a dynamic business environment. Unfortunately, adaptive systems are still difficult to build due to their high complexity. In this paper, we propose a new approach for developing highly adaptable Web services based on a synergy between context-aware Web service policies and Aspect Oriented Programming (AOP). This synergy is achieved through the elaboration of an innovative extension of the Web Service Policy Language (WSPL), which allows for context specification at both policy and rule levels. In addition, we provide a tool for the development of aspect-oriented policies, including an option to translate WSPL policies into aspect-oriented policies. These policies can be automatically woven into composite Web services (e.g., a BPEL process). The elaborated synergy between context, policies and aspects allows service providers to increase the level of adaptability of Web services at different layers of applications.

Keywords: Context-aware Policies, Aspects, Web Services, Business Process Execution Language.

1 Introduction

Web services are the key technology behind the tremendous widespread of e-Applications such as e-Business, e-Government and e-Science [5,12,19,20,27]. Although active research efforts and achievements in highly adaptable Web services are most desirable, still efforts need to be made in order to reach better adaptability. A Web service is called *adaptable* if it is able to change its behavior at runtime according to the changes in the execution environment. Policies can be a good means to

achieve such adaptability. In fact, during the last few years, managing Web services through the specification of policies has become one of the hot and active research areas [9,10,11,13,14,15,17,22,23,25,30]. A *Web service policy* is a set of rules that define the capabilities and requirements of a Web service. It also governs the runtime behavior, quality and result of a Web service. Two major languages were designed to specify these policies, namely the WS-Policy [7,8] and the Web Services Policy Language (WSPL) [6,14]. The first language is a W3C standard, while the second is based on the OASIS XACML standard [1].

We present in this paper a new approach to achieve highly adaptable Web services through context-aware Web service policies, which are not only *policy context* aware but also *rule context* aware. In fact, the integration of context is performed at both policy and rule levels. In particular, we extend WSPL to specify context for both policies and rules[1], which results in context-aware WSPL policies. Our target language is any XACML-based language and particularly we selected WSPL for its simplicity and low learning curve. This language has been used in [14,24,28] for specifying several kinds of policies such as business, security, and privacy policies. However, our proposed ideas can be easily applied to any XACML-based language.

We also develop a policy tool that provides a framework for the specification and implementation of aspect-oriented policies, including an option to translate WSPL policies into aspect-oriented policies, since WSPL policies cannot be directly applied to a composite Web service (e.g., a BPEL process). With this tool, we can handle highly adaptable composite Web services. It is worth noting that our context-aware Web service rules can be easily updated without any side-effect on the business logic of the BPEL process. This is a critical and useful feature in this fast changing world, as it adds an extra degree of flexibility in developing composite Web services.

The rest of the paper is organized as follows. Section 2 provides some background information about context, WSPL and AOP. It also introduces a scenario that will be used throughout the paper to illustrate our approach. Section 3 is dedicated to the presentation of some related works. Section 4 describes the new proposed approach focusing on how context can be specified for WSPL rules and applied to composite Web services through AOP weaving. Section 5 is devoted to the presentation of our aspect-oriented policy tool. Finally, Section 6 concludes the findings of the paper and draws future research directions.

2 Background

This section gives some background information about the concepts of context, WSPL and AOP as well as introduces our scenario, which is related to a flight system for the travel agency staff to book flights.

2.1 Context

Dey and Abowd defined context – which is widely adopted in the literature – as "*any information that can be used to characterize the situation of an entity. An entity is a*

[1] Note that current WSPL versions do not capture context at the rule level.

person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves" [3]. In this paper, we focus on Web service rule context, which refers to any information that affects a rule's guidance to the interaction between a Web service and its requester. For instance, let us assume a policy of a `flightBooking` Web service, which contains a discount rule that offers Gold members a 50% discount rate for special occasions and a 20% regular discount rate in other times. The "special" and "regular" occasions constitute the context of the discount rule because the occasions affect the discount rate that a Gold member receives.

2.2 Web Service Policy Language

WSPL is an XML language based on the OASIS XACML standard [1]. It is a kind of XACML profile for Web services. WSPL is known to be simple and to have a low learning curve for Web service policies specification. WSPL has three top-level elements: *PolicySet*, *Policy*, and *Rule*. A PolicySet is the container for policies. A policy is a sequence of one or more rules. Rules are listed in order of preference, with the most preferred choice listed first [14]. Each WSPL rule defines a constraint that the service needs to abide. Predefined constraint operators include: *equals*, *greater than*, *greater than or equal to*, *less than*, *less than or equal to*, *set-equals*, and *subset*. However, current WSPL does not capture context at rule level, which seriously limits the adaptability of WSPL policies.

2.3 Aspect-oriented Programming

The main objective of AOP is to have a clean separation between cross-cutting concerns. These are parts of a program that are not related to the program's primary function but tangled with (i.e., dependent on) other program functions. In other words, AOP aims to isolate supporting functions from the main program's business logic. This can be achieved through the definition of aspects. Each aspect is a separate module in which *pointcuts* are defined. A *pointcut* identifies one or more join points. A *join point* identifies one or more flow points (e.g., method calls) in a program (in our case a program is nothing but a BPEL process). At these join points, advices are to be executed. An *advice* contains some code that can alter the process behavior before, after or around a certain flow point, hence the advice can be called *before advice*, *after advice* or *around advice* respectively. The integration of aspects within an application code is called *weaving* and is performed based on one of the weaving technologies such as AspectJ [4].

2.4 The Flight System Scenario

To better illustrate our approach, we suggest a *flight system* as a running example. The system is composed of four Web services, a BPEL process and a graphical user interface that allows users to invoke the four Web services as depicted in Figure 1.

The flight system components are:

– *Financial Data*: a Web service that allows a staff member for example to request the revenues and expenses of the travel agency for a given month.

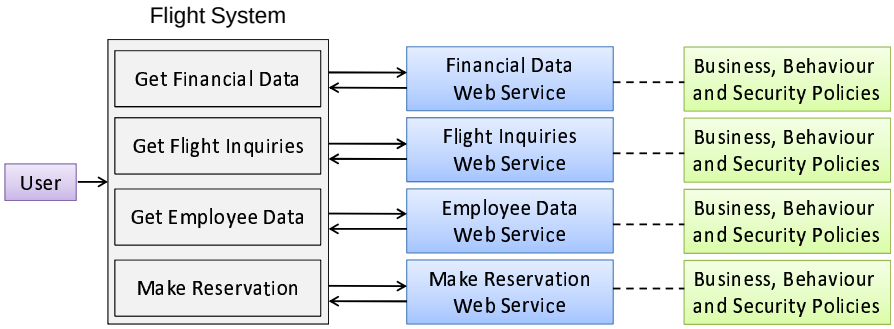

Fig. 1. Flight system components

- *Flight Inquiry*: a Web service, which returns a list of the available flights including their names, time tables (departure and arrival time and date), available seats and ticket prices.
- *Employee Information*: a Web service that allows the user to view staff information including the employee's full name, phone number, email address, post and his/her office number.
- *Make Reservation*: a Web service that enables the travel agent to reserve a seat on a certain flight.

Each staff member has an ID and a password stored in the database in addition to other personal information. Each time a user wishes to access any of the four flight system services, both the *Authentication* and *Access Control* policy rules may be applied depending on the context in which the Web service is running. These policy rules are weaved within the BPEL process of the flight system at specific points of execution. The technical details regarding the weaving of policy rules are provided later.

3 Related Work

Dynamic and adaptive systems are highly desirable and have been extensively investigated by researchers. Three major ingredients are commonly adopted to achieve such dynamism, namely context, business policy/rule and AOP. We review in what follows the research initiatives regarding the use of these ingredients to come out with adaptable and context-aware Web services.

Keidl and Kemper [9] explore the role of context in context-aware adaptable Web services. They mainly focus on the context of clients and Web services based on three automatic context processing components: Web services themselves, context plugins, and context services. In their framework, two main parts are required: a distributed infrastructure that transmits context between clients and Web services as well as manages context processing, and the supported context types which are extensible at any time. An added feature in our work compared to their work is that we leverage the AOP concept

to manage business policies and rule context and plug them into the BPEL specification of Web services composition.

Mrissa et al. [26] also investigate context and look into the way of achieving an automatic semantic interpretation of the heterogeneous data exchanged between the composed semantic Web services. They propose a context-based mediation approach for semantic Web services composition. In their approach, the semantic Web services are described with contextual details. A mediator Web service is automatically generated and invoked for each data flow, and the mediators use the contextual details to explicitly interpret the implicit assumptions (i.e., context) on the data flows. Our context definition is separated from the service code and is more related to policies and rules which guide the behavior of Web services. In our approach, service code changes and policy updates can be operated with minimum interference with each other.

Maamar et al. [22] investigate how policies and context can be used together in composing Web services. They propose a context-based policy approach for Web services composition, but their approach focus on how engagement, mediation and deployment policies can be used to manage the transactions between Web service bindings at component, composition, semantic and resource levels. They also investigate the issue of using the context to provide necessary environmental information to track the composition progress. Compared to their work, we add a level of flexibility to the composition using AOP. Our approach focuses on introducing context of business rules in WSPL extension and weaving the context-aware rules into business processes through AOP.

The work of Baresi and Guinea [16] is about weaving rules into a BPEL process for the sake of monitoring the execution of the process including data collection and validation. The monitoring rules can be dynamically selected and executed at run-time to achieve continuous monitoring of the running process. Our weaving is for context-based rules, which allows having a high level of adaptation for Web services. Besides, the adaption can be at several levels: business logic, security, etc., and not just only to the monitoring of the execution.

Yu et al. [30] investigate the role of aspects and rules in adaptive systems, but their approach is different from ours. Their model-driven approach divides the functionality of a system into a stable part (e.g., a business process) and a volatile part (e.g., a business rule) which are described respectively by a base model and a variable model. The volatile part is weaved into the stable part through aspect-oriented approach. Such a system needs to run on the top of a process engine and a rule engine. Although the two models can be evolved independently and the stable part can be adaptable to the volatile part, no adaption at volatile part is even mentioned. Our approach allows context adaption at both business process level through relevant business policies and at business rule level through rule context. Therefore our approach adds extra level of flexibility and adaptability.

The work of Courbis and Finkelstein [18] is also related to aspects. They illustrate how unforseen features (requirements or context) can be dynamically adapted to Web services-based and aspect-oriented applications through Domain-Specific Aspect Languages that allow aspect weaving not only at BPEL process level but also at BPEL engine and semantic analyzer levels. Beside the aspect concept, our adaptation relies on two other ingredients: context and policies, which makes the service highly adaptable.

Besides, we build our approach on top of known policy languages so there is no need for extra domain specific languages.

Charfi and Mezini [21] focus on the design and implementation of *AO4BPEL*, which extends BPEL to support aspect features and provides a framework to define workflow aspects for cross-cutting concerns such as logging and security of a business process. AO4BPEL can dynamically change the deployed process by simply activating or deactivating the defined aspects. Although AO4BPEL allows dynamic and modular Web service composition, it does not support dynamic rule adaption since all the rules are part of the process. In our approach, business rules are specified as WSPL policies which are translated into aspects and weaved into the core business process. Furthermore, we extend WSPL to introduce context for the rules so that they are context-aware, adaptable and can be updated without affecting the business process. This strategy allows minimizing the runtime cost of the activation and deactivation of aspects.

Agarwal and Jalote [29] propose yet another different context approach for dynamically adapting Web service composition based on users' Non-Functional Requirements (NFRs). In their approach, NFRs for each partner service of a BPEL process is specified and used by a system to dynamically select suitable services at run-time. The system can be integrated within a BPEL engine by an aspect-oriented approach. Our approach can capture such users' non-functional requirements and even business requirements related to the business logic. They can be captured through business policy rules that dynamically guide the invoking of the required services at run-time. The policy rules are weaved into a BPEL process based on AOP.

In one of the most recent related works, Li et al. [31] design a semantic-based weaving mechanism for context-aware Web service composition. In their approach, contexts are defined as ontology concepts and are statically weaved with services as aspects, which allows the service to be semantically adaptable to different applications without modifying the service execution engine. A common point between this work and ours is that we adopt also static weaving. However, besides the use of context and aspects, our approach leverages policies, which adds a level of expressiveness to the injected context-aware policies.

Our adaptability approach spans over three ingredients: Context, Policies, and AOP. Policies are endowed with context at policy and rule levels. This increases the level of adaptability of Web services without the need for an extra ontology to capture such context. The elaborated context-aware policies are translated into aspects and woven statically into the BPEL process of the composite service. Such static integration allows avoiding additional runtime overhead as proposed in several research initiatives.

4 The Proposed Approach

This section describes our approach in details, focusing mainly on specifying context for rules based on a WSPL extension and applying context-adaptable WSPL policies to composite Web services through AOP weaving.

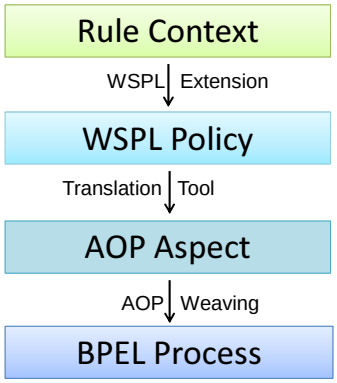

Fig. 2. An overview of the proposed approach

4.1 An Overview

As shown in Figure 2, our approach involves four entities and three operational methods. The four entities are: *Rule Context*, *WSPL Policy*, *AOP Aspect*, and *BPEL Process*. The three operational methods are: *Context Specification*, *Policy Translation*, and *Policy Integration*. The Context Specification method includes operations for specifying a Web service policy context such as the WSPL Extension that we propose to specify a rule context for WSPL policies. The Policy Translation method comprehends operations for translating an exiting policy to a required format such as the Translation Tool that is used to translate a WSPL policy to an AOP aspect in this work. Finally, the Policy Integration method contains operations for applying context-based policies to Web services such as AOP Weaving that weaves an AOP aspect into a BPEL process.

4.2 Specifying Context for WSPL Rules

For the flight system that is described in Section 2.4, we define the following three levels of Web service policies to show how context can be specified in WSPL rules:

- The *Business* policy that identifies the rules that govern the operations of a Web service (e.g., the discount rate for a certain occasion),
- The *Behavior* policy that defines the quality attributes of a Web service (e.g., the response time for a service request), and
- The *Security* policy that describes the rules of security measurement and algorithms (e.g., the authentication algorithm for a certain interaction).

The detailed policies, rules and their respective contexts are specified in the pseudo code shown in Figure 3. The first part of the figure (left-hand side) is related to a business policy. This policy includes a discount rule that offers the Gold members a 50% *special* discount rate in special occasion but a 20% *regular* rate in other times. In other words, the Gold members get either a *special* or a *regular* discount rate depending on

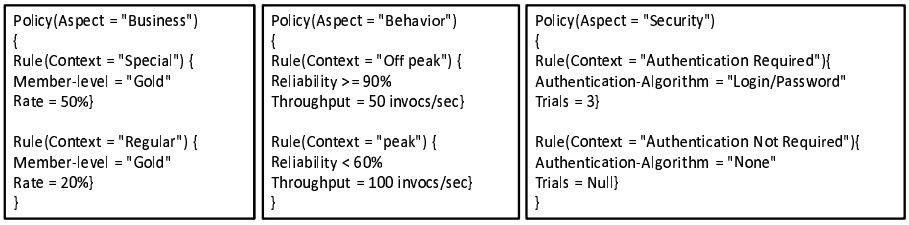

Fig. 3. Context for business, behavior and security policies

the occasions when they make the booking. The middle part of Figure 3 describes a behavior policy that depends on two contexts: *Peak* and *Off Peak* time. In an Off Peak time, the reliability of the Web service is more than 90% when the service request is 50 invocations per second. On the other hand, in a Peak time, the reliability of the Web service is less than 60% when the service request is 100 invocations per second. The last part of Figure 3 provides a security policy that again depends on two contexts: *Authentication Required* and *Authentication Not Required*. If an authentication is required, the user needs to provide a correct password in no more than three trials in order to access the flight system. If the authentication is not required, the user can invoke a service without being asked for any authentication.

4.3 Context-adaptable WSPL Policies in Composite Web Services

A policy would be useless if it could not be applied to its corresponding Web service. Unfortunately there exists no straight forward way to apply WSPL policies directly to a composite Web service. Our approach overcomes this problem by establishing a synergy between context-aware policies and Aspects. This synergy is detailed in what follows.

Extracting AOP Aspects Our WSPL extension for the specification of context-aware policies is a paramount ingredient of our approach. Indeed, it is a step towards achieving a separation between business, behavior, and security concerns. The way context-aware policies are specified in this work allows to smoothly transform each context-based WSPL policy into a corresponding aspect. The separation of concerns allows better accommodation of changes in Web service policies. In fact, any change in a policy can be smoothly achieved through the update of the related context component in the aspect file without any side-effect on the business logic of the Web service. In other words, there is no need to update the business logic of a Web service. This capability adds an extra degree of flexibility in developing composite Web services.

Figure 4 outlines the BPEL process of the flight system. This process can invoke any of the four component Web services of the flight system taking into consideration its business, behavior and security policies. These policies are weaved within the BPEL process in order to frame the execution of the invoked Web service. The weaving of such policies relies on an aspect-oriented technology.

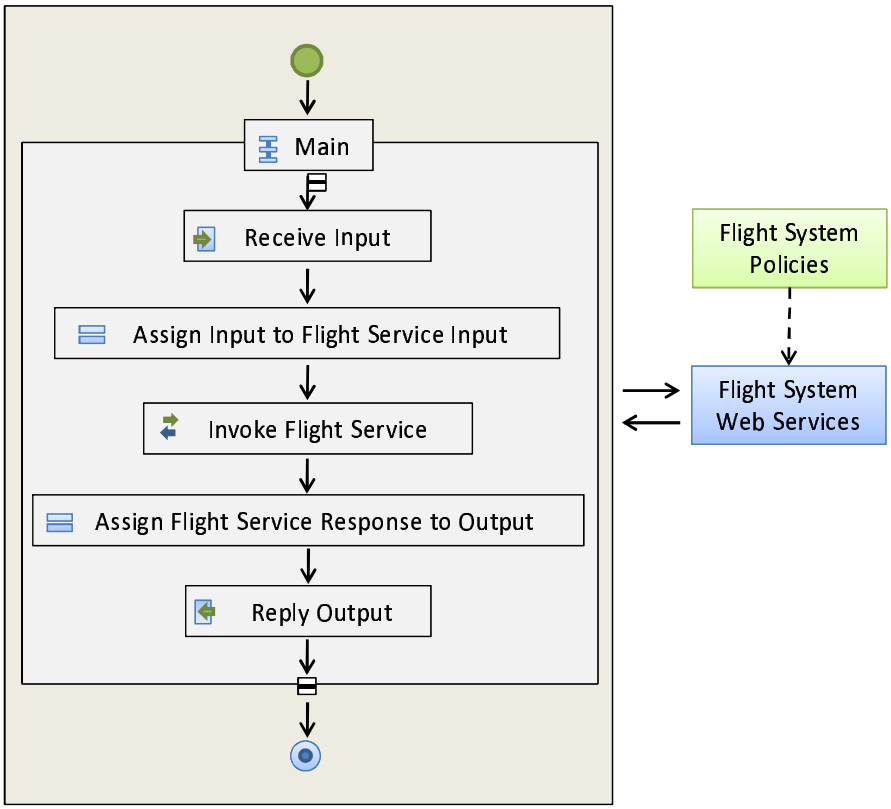

Fig. 4. Flight system BPEL process

Figure 5 outlines the weaving of business, behavioral and security aspects into the BPEL process of the flight system. This process can invoke any of the four flight system Web services with its three related aspects. The idea is to translate all three policies into aspects and then to perform a merging of these three aspects with the main BPEL process. We illustrate the translation process based on our flight system scenario, which includes three context-aware policies that are transformed into three aspects as explained in what follows.

User Authentication BPEL Aspect Listing 1.1 illustrates the context-driven authentication aspect, which is extracted from the security policy that is defined in Figure 3. An authentication procedure (e.g., entering a password) may or may not be required when a user invokes a Web service of the flight system, depending on the context value (Authentication Required or Authentication Not Required), this condition is reflected in the If activity in line 8. Such context is extracted from the aforementioned security pol-

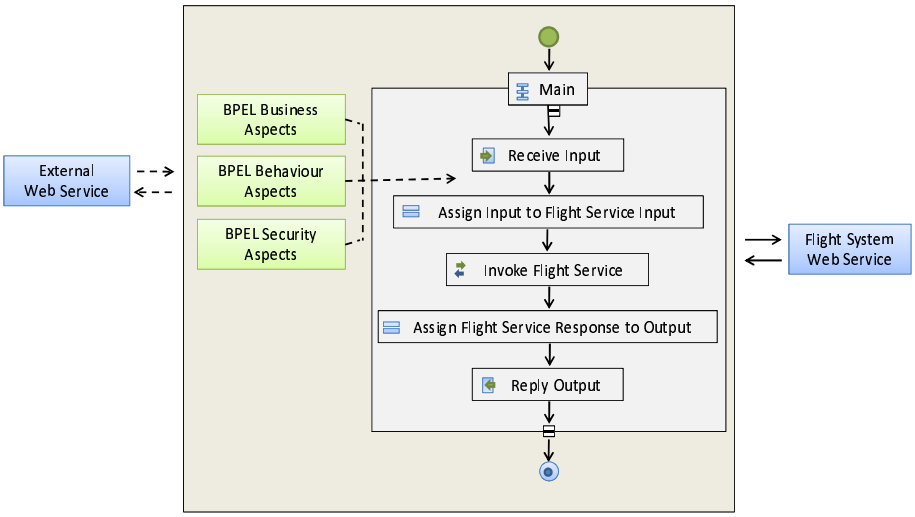

Fig. 5. Weaving the three context-based aspects with the flight system BPEL process

icy. The authentication module is endowed with a Role Based Access Control (RBAC) model. The objective of the authentication module is to assign each user a role: either *Director*, *Manager* or *Staff*, and the corresponding role permission(s) within the flight system Web services. For example, a Director can invoke all four services. The pointcut $P1$ in line 3, is the <bpel:assign...> BPEL construct that is responsible of assigning the input to the requested Web service and consequently invoking it.

The BPEL authentication verification code is injected before the execution of the *assign* construct. The authentication verification code first assigns the user's login information to the Authentication Web service request message, as depicted from line 13 till line 23. Afterwards, in line 26, the Authentication Web service is invoked. The invoked Web service calls the *UserAuthentication* operation that loops through the database and returns one of the following four possible indices: the username is invalid, the username and password are correct, the username is valid but the password is incorrect, and trials run out (the flight system gives the user a maximum of three login trials).

If the authentication fails, the Web service is invoked again to get the appropriate error string for the returned index, as shown from line 30 till line 65. The returned error string is then assigned to the BPEL process output variable and forwarded to the user. On the other hand, if the authentication succeeds, the BPEL process continues its execution by invoking the requested Web service. All the aforementioned verification steps are represented graphically in Figure 6, which depicts a flow-based graphical representation[2] of the flight system BPEL process after weaving it with the security aspect.

[2] This representation is automatically generated by the Eclipse BPEL Designer plugin

Listing 1.1. Aspect for Authentication

```
1. Aspect Authentication
2. {
3. Pointcut P1: <bpel:assign validate="no" name="Assign MenuIndex and Rate To
     FlightSystem">;
4.
5. Before P1
6. {
7. <!-Check if Authentication is Needed -->
8. <bpel:if name="IsAuthenticationNeeded">
9. <bpel:condition><![CDATA[($input.payload/tns:IsUserTrusted="NotTrusted")]]></
     bpel:condition>
10.
11. <bpel:sequence>
12. <!-Initialize Authentication Request Message-->
13. <bpel:assign validate="no" name="Assign Input to Authentication message">
14. <bpel:copy>
15. ...
16. </bpel:copy>
17. <!--Copy from Input Message Username and Password to AuthenticationRequest
     Message-->
18. <bpel:copy>
19. <bpel:from part="payload" variable="input">
20. <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
21. ...
22. </bpel:copy>
23. </bpel:assign>
24.
25. <!-Invoke Authentication Web Service-->
26. <bpel:invoke name="Invoke Authentication" partnerLink="AuthenticationPartner"
     operation="userAuthentication" portType="ns:Authentication"
27. inputVariable="AuthenticationPartnerRequest" outputVariable="
     AuthenticationPartnerResponse">
28. </bpel:invoke>
29.
30. <!-Check if User is Authenticated-->
31. <bpel:if name="If Invalid User">
32. <bpel:condition><![CDATA[($AuthenticationPartnerResponse.parameters/ns:
     userAuthenticationReturn!=2)]]>
33. </bpel:condition>
34.
35. <bpel:sequence>
36. <!-Initialize GetErrorStr Request Message-->
37. <bpel:assign validate="no" name="Assign AuthenticationResponse to GetErrorStr
     ">
38. ...
39. <!-- Assign the Authentication response to the GetErrorStr Request Message-->
40. ...
41. </bpel:assign>
42.
43. <!-Invoke the GetErrorStr Web Service-->
44. <bpel:invoke name="Invoke GetErrorStr" partnerLink="AuthenticationPartner"
     ...">
45. </bpel:invoke>
46. <bpel:assign validate="no" name="Assign GetErrorStr To Output">
47.
48. <!--Initialize the BPEL output Variable-->
49. <bpel:copy>
50. ...
51. <bpel:to variable="output" part="payload"></bpel:to>
52. </bpel:copy>
53. <!-Copy from GetErrorStr Response message to the BPEL Output Message-->
54. <bpel:copy>
55. ...
56. </bpel:copy>
57. </bpel:assign>
58.
59. <!-Return Error String in case of Invalid User-->
60. <bpel:reply name="Return ErrorString" partnerLink="client" operation="process"
      portType="tns:BPELProcess" variable="output">
61. </bpel:reply>
62. </bpel:sequence>
63. </bpel:if>
64. </bpel:sequence>
65. </bpel:if>
66. }
```

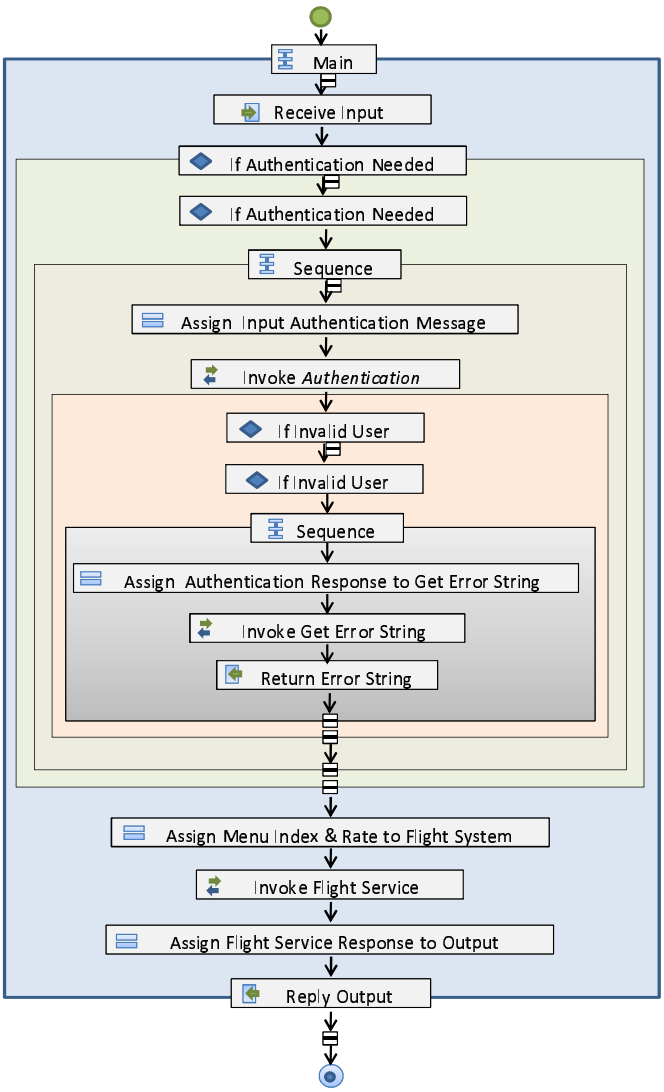

Fig. 6. Authentication BPEL process chart

Business BPEL Aspect Listing 1.2 depicts the context-driven business aspect, which is extracted from the business policy that is presented in Figure 3. The checking of the context value (discount or regular) is performed in order to set the accurate discount rate. Such context is extracted from the aforementioned business policy. The pointcut $P2$ in line 3, is the <bpel:assign...> BPEL construct that is responsible of assigning the input to the requested Web service and consequently of invoking it.

Listing 1.2. Aspect for Business

```
1. Aspect Business
2. {
3. Pointcut P2: <bpel:assign validate="no" name="Assign MenuIndex and Rate To
     FlightSystem">;
4.
5. Before P2
6. {
7. <!--Check the User's Member Level and assign a discount rate respectively-->
8. <bpel:if name="Check MemberLevel">
9.
10. <Check if Context is discount and the user's member level is gold-->
11. <bpel:condition><![CDATA[(($input.payload/tns:Context="Discount")and($input.
     payload/tns:MemberLevel="Gold"))]]>
12. </bpel:condition>
13.
14. <!--Initialize BPEL Input Message-->
15. <bpel:assign validate="no" name="Assign Rate">
16. <bpel:copy>
17. ...
18. </bpel:copy>
19.
20. <!--Assign Discount Rate to 50-->
21. <bpel:copy>
22. <bpel:from>
23. <![CDATA["50"]]>
24. </bpel:from>
25. <bpel:to part="payload" variable="input">
26. <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
27. <![CDATA[tns:Rate]]></bpel:query>
28. </bpel:to>
29. </bpel:copy>
30. </bpel:assign>
31.
32. <Check if Context is regular and the user's member level is gold-->
33. <bpel:elseif>
34. <bpel:condition><![CDATA[(($input.payload/tns:Context="Regular")and($input.
     payload/tns:MemberLevel="Gold"))]]>
35. </bpel:condition>
36.
37. <!-Initialize the BPEL Input Message-->
38. <bpel:assign validate="no" name="Assign Rate">
39. <bpel:copy>
40. ...
41. </bpel:copy>
42.
43. <!-Assign Discount Rate to 20-->
44. <bpel:copy>
45. <bpel:from>
46. <![CDATA["20"]]>
47. ...
48. </bpel:copy>
49. </bpel:assign>
50. </bpel:elseif>
51. </bpel:if>
52. }
```

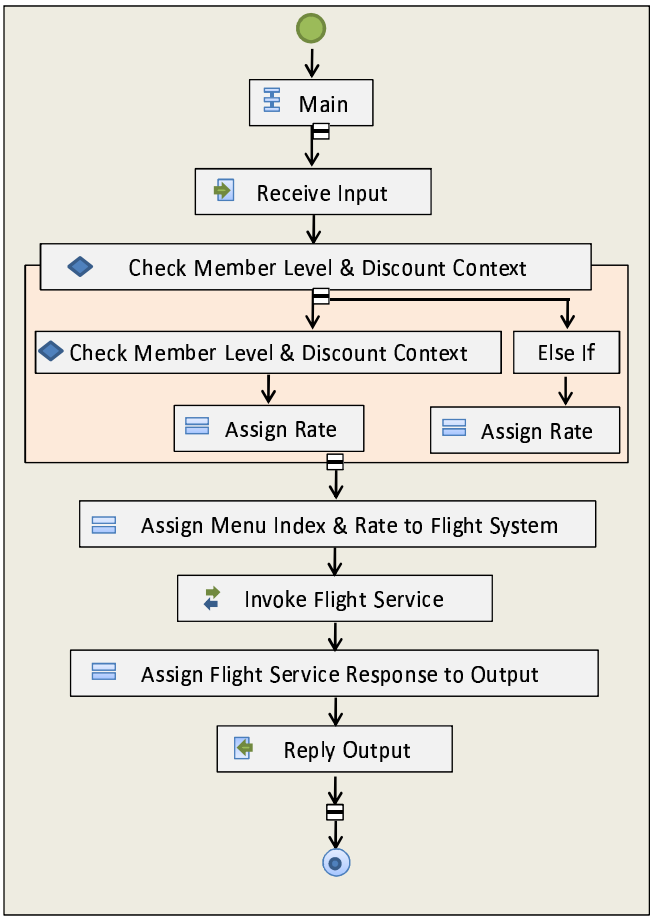

Fig. 7. Business BPEL process chart

The BPEL business verification code is injected before the execution of the *assign* construct. First, the business verification code integrates an IF/Else condition to check the values of the context (i.e. discount or regular) and the member level of the user, depicted in line 8. If the context is discount and the member level is gold, assign a value 50 to the discount rate, as seen from line 10 to 31. Else, if the context is regular and the member level is gold, assign a value 20 to the discount rate, as seen from line 32 to 51. Afterwards, the BPEL process continues its execution to invoke the requested Web service. A flow-based graphical representation of the flight system BPEL process after weaving it with the business aspect is depicted in Figure 7.

Behavioral BPEL Aspect Listing 1.3 depicts the context-driven behavioral aspect, which is extracted from the behavior policy that is depicted in Figure 3. The checking of the context value (On-peak or Off-peak) is performed in order to check that the current request does not make the number of allowed connections exceed the maximum. Such context is extracted from the aforementioned behavior policy. The pointcut $P3$ in line 3, is the <bpel:assign...> BPEL construct that is responsible of assigning the input to the requested Web service and consequently invoking it.

The BPEL behavior verification code is injected before the execution of the *assign* construct. The behavior verification code first integrates an IF/Else condition to check the values of the context (i.e. On-peak or Off-peak) and the number of requests. If the context is On-peak (line 34 to 56) and the number of requests is greater than 100, return an exceeding limit error message to the user. If it is Off-peak (line 8) and the number of requests is greater than 60, return an exceeding limit error message to the user (line 29). Otherwise, the BPEL process continues its execution by invoking the requested Web service. A flow-based graphical representation of the flight system BPEL process after weaving it with the behavior aspect is depicted in Figure 8.

5 Aspect-oriented Policy Specification Tool

We have successfully implemented an aspect-oriented policy specification tool, which constitutes a framework for developing aspect-oriented policies for Web services. This tool includes a menu option to load and compile a context-based WSPL policy as depicted in Figure 9. The tool also verifies the syntactic correctness of a WSPL policy before translating it into an aspect that is shown in Figure 10. The generated aspect can be automatically weaved into a BPEL file of a composite Web service by simply selecting the `Weave Aspect` menu option (see Figure 10).

Beside the weaving capability, there are other functionalities provided by this tool such as the detection of possible coding errors in user defined policies or aspects. Under development is another capability of merging two context-based WSPL policies of atomic Web services into a common policy which is then translated into an aspect and weaved automatically into the BPEL code.

5.1 Aspect-oriented Policy Framework

Our framework, which is illustrated in Figures 9 and 10, offers a user friendly environment to perform a binding between a WSPL policy and a BPEL process. It allows the automatic transformation of a WSPL policy into a BPEL aspect, which is then statically weaved into the specified process. To perform the weaving process, the user should first go to the “Policies” menu and select the “New WSPL” menu item. A new editor pane appears under the “Policies” panel where the WSPL policy is developed or loaded. Afterwards, the user must click on the “Generate Aspect” menu item to turn the WSPL code into a BPEL aspect. On click, a dialog box appears, requesting from the user to specify to which BPEL process the policy corresponds. This is needed because the policy should be bound to a specific BPEL process. Once the BPEL aspect is generated, the user should go to the “BPEL Weaving” menu and select the “Weave Aspect” menu

Listing 1.3. Aspect for Behavior

```
1. Aspect Behavior
2. {
3. Pointcut P3: <bpel:assign validate="no" name="Assign MenuIndex and Rate To
     FlightSystem">;
4.
5. Before P3
6. {
7. <!-Check if Number of Requests Exceeds Limit-->
8. <bpel:if name="Check NumberOfRequest">
9.
10. <!-Off Peak time and Number of Requests Exceeds 60-->
11. <bpel:condition><![CDATA[(($input.payload/tns:PeakContext="Off Peak")and(
     $input.payload/tns:NumberOfRequest>60))]]></bpel:condition>
12. <bpel:sequence>
13.
14. <!-Initialize BPEL Output Message-->
15. <bpel:assign validate="no" name="Assign ErrorMessage">
16. <bpel:copy>
17. ...
18. </bpel:copy>
19.
20. <!-Reply With Error Message-->
21. <bpel:copy>
22. <bpel:from>
23. <![CDATA["Number of Requests exceeds the limit"]]>
24. ...
25. </bpel:copy>
26. </bpel:assign>
27.
28. <!-Return Error to Client-->
29. <bpel:reply name="Reply with ErrorString" partnerLink="client" operation="
     process" variable="output"></bpel:reply>
30. </bpel:sequence>
31.
32. <bpel:elseif>
33. <!-On Peak time, and Number of Requests Exceeds 100-->
34. <bpel:condition><![CDATA[(($input.payload/tns:PeakContext="Peak")and($input.
     payload/tns:NumberOfRequest>100))]]>
35. </bpel:condition>
36. <bpel:sequence>
37.
38. <!--Initialize BPEL Output message-->
39. <bpel:assign validate="no" name="Assign Error Message">
40. <bpel:copy>
41. ...
42. </bpel:copy>
43.
44. <!-Reply with Error Message-->
45. <bpel:copy>
46. <bpel:from>
47. <![CDATA["Number of Requests exceeds the limit"]]>
48. ...
49. </bpel:copy>
50. </bpel:assign>
51.
52. <!-Send error message to client-->
53. <bpel:reply name="Reply with ErrorString" partnerLink="client" operation="
     process" variable="output"></bpel:reply>
54. </bpel:sequence>
55. </bpel:elseif>
56. </bpel:if>
57. }
```

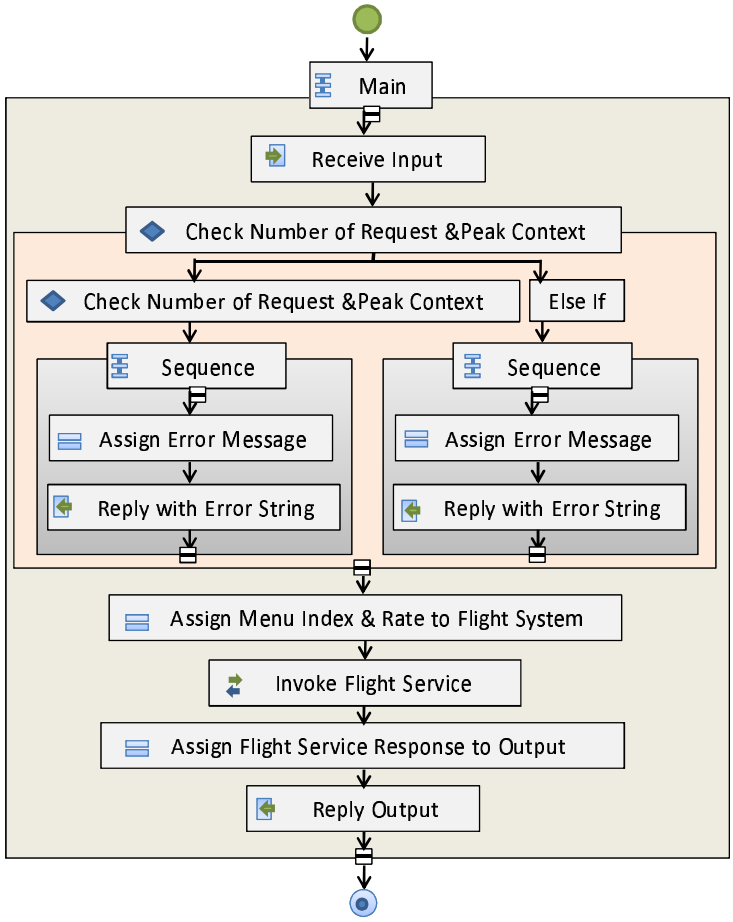

Fig. 8. Behavior BPEL process chart

item. A “Weaving Completed” message then appears indicating that the weaving was successfully performed.

In what follows, we explain in details the inner-working of two major components of our tool: 1) The WSPL-AspectBPEL generator that allows the standard description of WSPL policies and the automatic generation of their corresponding BPEL aspects, and 2) The AspectBPEL weaver that performs the static weaving of context-aware policies into the BPEL code.

5.2 WSPL-AspectBPEL Generator

Our WSPL parser is built on top of the Java-based DOM parser for XML. It takes as input a context-aware policy and provides as output an aspect. It begins by parsing each

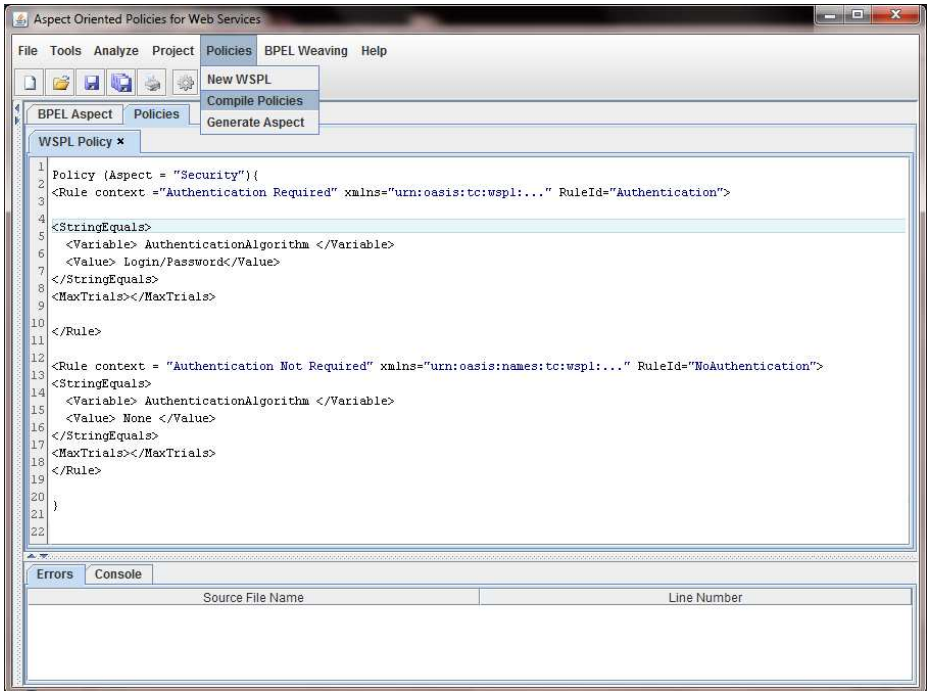

Fig. 9. Context-based WSPL policy specification

policy rule name to identify if the rule effect is a *permit* or a *deny*. Afterwards, the parser gets all the identified resources and actions presented in the policy. Each action on a resource is matched to an invoke activity in the BPEL process side and a location identifier is generated for each resource action. For each resource, a loop runs on the list of specified actions. It then scans each action via a second parser to search within the selected process code and returns the name of the invoke activity, which has as port type the resource attribute value and as operation the action attribute value.

A generated aspect may contain as many location identifiers as the number of returned invoke activities. For each location identifier, a location behavior is defined based on the subject's tag. The information extracted from the subject's tag is wrapped within an *If* activity, where the condition, written in XPATH language, is expressed as follows: the attribute id, followed by the *MatchId* then the attribute value. For example, the *SubjectMatch* condition that is presented in Listing 1.4 will be matched if the *subject-id* is equal to bob.

The rule effect plays a significant role in determining the *MatchId* value. If the rule effect is a *deny*, the *MatchId* is kept as it is. Otherwise, if the rule effect is a *permit*, then the *MatchId* function is complemented when matched into the XPATH condition. The *If* condition is a break point that determines whether the process gets to continue its predefined path, or halts because the conditions were satisfied. In other words, a rule

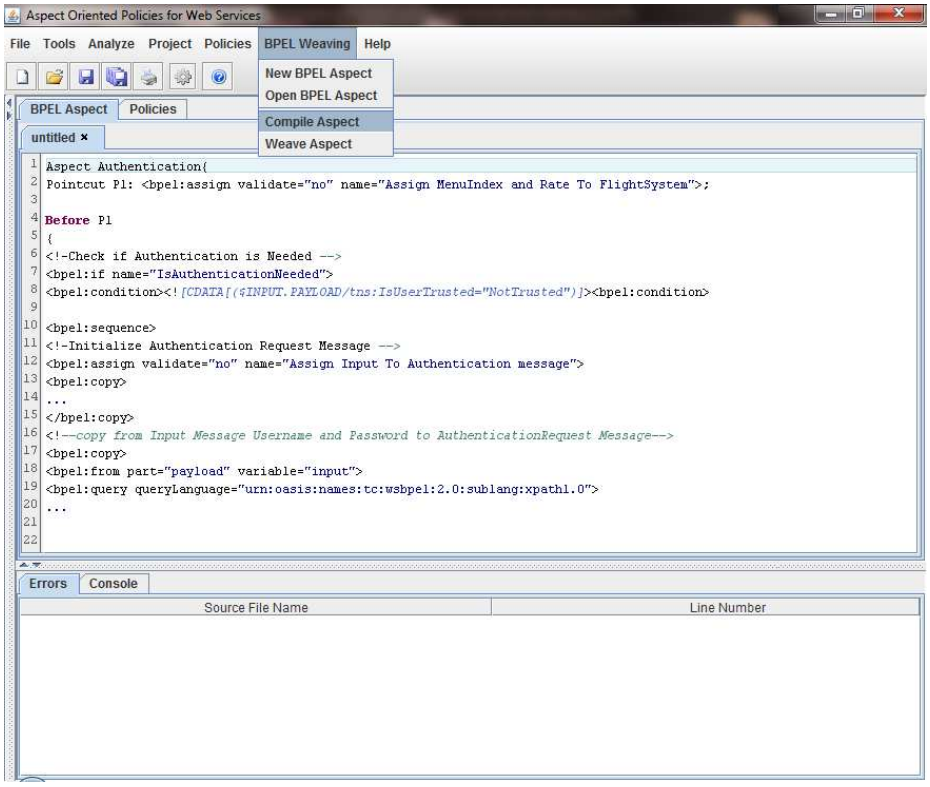

Fig. 10. Authentication aspect

with a *permit* effect and a subject tag that states that the username should be equal to bob is matched into an If statement. If the username is different from bob, it returns an error to the client. The list of location identifiers are then wrapped between a *BeginAspect* and an *EndAspect* statements.

5.3 AspectBPEL Weaver

In order to weave the generated aspect into a specific BPEL process, we have elaborated a language called *AspectBPEL* and developed its corresponding framework and weaver. The weaver integrates *AspectBPEL* aspects, that are either generated from policies or written by the developer, within the main BPEL process. It starts by identifying in the generated aspect the BPEL insertion point (i.e. *Before*, *After* or *Replace*). Afterwards, it takes as user input the BPEL location identifier (e.g. *assign*, *invoke*, *reply*, *receive*, *if*, etc.) and its signature, which represents the activity where the aspect code has to be inserted. Such input is used to identify the target join points in the main BPEL process. To perform such identification, a BPEL parser that is based on the *Apache-ode* library [2] is integrated in the *AspectBPEL* framework. Once appropriate joint points

Listing 1.4. XACML-based Access Control Policy Snippet

```
<SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema
                              string">\textbf{bob}</AttributeValue>
                        <SubjectAttributeDesignator
                            AttributeId="urn:oasis:names:tc:xacml:1.0:subject:
                                 subject-id"
                            DataType="http://www.w3.org/2001/XMLSchema string" />
</SubjectMatch>
```

are found, the aspect code is inserted in the original BPEL code and a message in the output console appears indicating that the weaving was successfully done.

6 Conclusion & Future Work

We proposed in this paper a new approach to achieve highly adaptable Web services. In particular, we extended WSPL to allow context-based specification of policy rules. Furthermore, we established a synergy between context and aspect concepts in order to integrate context-aware WSPL policies within composite Web services. The proposed approach is embedded in an *aspect-oriented policy* tool, which allows service providers to specify, compile, and weave context-aware policies into the BPEL code of a composite Web service. Our approach does not introduce any additional runtime overhead since the adaptability is performed at the static level. Besides, it allows the integration of very expressive context-aware policies into composite Web services.

Several future research directions arise. For instance, the context can be used as an interesting element in enforcing the security of a Web service and enhancing its performance. Another interesting issue is the use of context in matching user preferences and Web service capabilities. Indeed, our approach can be leveraged to establish a context-driven discovery protocol. Such protocol would allow a better matching between user requirements and Web service capabilities. The elaborated protocol can be also used to perform a fine-grained ranking of Web services.

References

1. T. Moses. OASIS eXtensible Access Control Markup Language (XACML), OASIS Standard 2.0. `http://www.oasis-open.org/committees/xacml/`.
2. Jarvana Search. Overview about Apache ode. `http://www.jarvana.com/jarvana/view/org/apache/ode/apache-ode-docs/1.2/apache-ode-docs-1.2.zip!/javadoc/overview-summary.html`.
3. G. Abowd, A. Dey, P. Brown, N. Davis, M. Smith, and P. Steggles. Towards a Better Understanding of Context and Context-Awareness. In *HUC 99: Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, pages 304–307, Karlsruhe, Germany, 1999. Springer-Verlag.
4. G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An Overview of AspectJ. In *Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP 01)*, pages 327–353, London, UK, 2001. Springer-Verlag.

5. M.P. Papazoglou. Web services and business transactions. *World Wide Web*, 6(1):49–91, 2003.
6. A. Anderson. An Introduction to the Web Services Policy Language (WSPL). In *POLICY 04: Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 189–192, Washington, DC, USA, 2004. IEEE Computer Society.
7. P. Nolan. Understand WS-Policy processing. Technical report, IBM Corporation, 2004.
8. J. Schlimmer. Web Services Policy Framework (WS-Policy), 2004. http://www-128.ibm.com/developerworks/webservices/library/specification/ws-polfram/.
9. M. Keidl and A. Kemper. Towards context-aware adaptable Web services. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 55–65. ACM, 2004.
10. H. Wang, S. Jha, M. Livny, and P. McDaniel. Security Policy Reconciliation in Distributed Computing Environments. In *Proceedings of the 5th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2004) in conjunction with The 9th ACM Symposium on Access Control Models and Technologies (SACMAT 2004)*, pages 137–146, Yorktown Heights, NY, USA, 2004.
11. Q. Z. Sheng and B. Benatallah. ContextUML: a UML-based modeling language for model-driven development of context-aware web services. In *Proceedings of the International Conference on Mobile Business (ICMB 2005)*, pages 206–212. IEEE, 2005.
12. B. Schmit and S. Dustdar. Model-driven development of Web service transactions. *Enterprise Modelling and Information Systems Architectures*, 1(1):46–55, 2005.
13. S. Agarwal and B. Sprick. Specification of Access Control and Certification Policies for Semantic Web Services. In *Proceedings of the 6th International Conference on Electronic Commerce and Web Technologies (EC-Web 2005)*, pages 348–357, Copenhagen, Denmark, 2005.
14. A. Anderson. Predicates for Boolean Web Service Policy Language (WSPL). In *Proceedings of The International Workshop on Policy Management for The Web (PM4W 2005) held in conjunction with The Fourteenth International World Wide Web Conference (WWW 2005)*, pages 53–55, Chiba, Japan, 2005.
15. A. Charfi and M. Mezini. Using Aspects for Security Engineering of Web Service Compositions. In *Proceedings of the Internationa Conference on Web Services (ICWS 05)*, pages 59–66, Orlando, USA, 2005.
16. L. Baresi and Guinea. S. Dynamo: Dynamic Monitoring of WS-BPEL Processes. In *Proceedings of the 3rd International Conference on Service Oriented Computing (ICSOC 2005)*, pages 478–483. Springer, 2005.
17. F. Clemente, G. Pérez, J. Blaya, and A. Skarmeta. Representing Security Policies in Web Information Systems. In *Proceedings of the Fourthen International World Wide Web Conference (WWW 2005)*, pages 1–6, Chiba, Japan, 2005.
18. C. Courbis and A. Finkelstein. Towards aspect weaving applications. In *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005)*, pages 69–77. ACM, 2005.
19. W.J.V. Den Heuvel, K. Leune, and M.P. Papazoglou. EFSOC: A layered framework for developing secure interactions between Web services. *Distributed and Parallel Databases*, 18(2):115–145, 2005.
20. OASIS Web Services Business Process Execution Language (WSBPEL) Technical Committee. Web Services Business Process Execution Language Version 2.0. Standard proposed by OASIS, 2007.
21. A. Charfi and M. Mezini. Ao4bpel: An aspect-oriented extension to bpel. *World Wide Web*, 10:309–344, September 2007.

22. Z. Maamar, D. Benslimane, P. Thiran, C. Ghedira, S. Dustdar, and S. Sattanathan. Towards a context-based multi-type policy approach for Web services composition. *Data & Knowledge Engineering*, 62(2):327–351, 2007.
23. Z. Maamar, N. Narendra, D. Benslimane, and S. Sattanathan. Policies for Context-driven Transactional Web Services. In *Proceedings of the 19th International Conference on Advanced Information Systems (CAiSE 2007)*, pages 249–263, Trondheim, Norway, 2007.
24. Z. Maamar, Q. Z. Sheng, H. Yahyaoui, D. Benslimane, and F. Liu. On Checking the Compatibility of Web Services' Policies. In *Proceedings of the International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2007)*, pages 125–130, Adelaide, Australia, 2007.
25. B. Medjahed and Y. Atif. Context-based matching for Web service composition. *Distributed and Parallel Databases*, 21(1):5–37, 2007.
26. M. Mrissa, C. Ghedira, D. Benslimane, Z. Maamar, F. Rosenberg, and S. Dustdar. A context-based mediation approach to compose semantic Web services. *ACM Transactions on Internet Technology*, 8(1):1–23, 2007.
27. B. Di Martino. Semantic Web services discovery based on structural ontology matching. *International Journal of Web and Grid Services*, 5(1):46–65, 2009.
28. Q. Z. Sheng, J. Yu, Z. Maamar, W. Jiang, and X. Li. Compatibility Checking of Heterogeneous Web Service Policies Using VDM++. In *Proceedings of the IEEE Workshop on Software and Services Maintenance and Management (SSMM 2009) held in conjunction the 2009 IEEE Congress on Services, Part I (SERVICES I 2009)*, pages 821–828, 2009.
29. V. Agarwal and P. Jalote. From specification to adaptation: An integrated qos-driven approach for dynamic adaptation of Web service compositions. In *Proceedings of the 8th IEEE International Conference on Web Servic (ICWS 2010)*, pages 275–282. IEEE Computer Society, 2010.
30. J. Yu, Q. Z. Sheng, and J. Swee. Model-Driven Development of Adaptive Service-Based Systems with Aspects and Rules. In *Proceedings of Web Information Systems Engineering (WISE 2010)*, pages 548–563, Springer, 2010.
31. L. Li, D. Liu, and A. Bouguettaya. Semantic based aspect-oriented programming for context-aware Web service composition. *Information Systems*, 36:551–564, 2011.