Towards Behavioral Web Services Using Policies

Zakaria Maamar  
*Zayed University, Dubai, U.A.E*

Djamal Benslimane  
*Claude Bernard Lyon 1 University, Lyon, France*

Ghita Kouadri Mostéfaoui  
*Oxford University, Oxford, UK*

Sattanathan Subramanian  
*University of Namur, Namur, Belgium*

Qusay H. Mahmoud  
*University of Guelph, Guelph, Canada*

Abstract

Making Web services context-aware is a challenge. This is like making Web service expose appropriate behaviors in response to changes detected in the environment. Context awareness requires a review and extension of the current execution model of Web services. This paper discusses the seamless combination of context and policy to manage behaviors that Web services expose during composition and in response to changes in the environment. For this purpose, a 4-layer approach is devised. These layers are denoted by policy, user, Web service, and resource. In this approach, behavior management and binding are subject to executing policies of types permission, obligation, restriction, and dispensation. A prototype that illustrates how context and policy are woven into Web services composition scenarios is presented, as well.

**Keywords.** Behavior, Composition, Policy, and Web service.

1 Introduction

Web services are among the promising technologies for integrating business processes despite of distribution and heterogeneity constraints. This primarily happens because of the possibility of composing Web services according to different specifications that reflect the way these business processes are performed (e.g., car rental). However, despite this promising future through different standards and research & industrial initiatives [6, 15, 29, 36, 37], Web services still lack the capabilities that could make them match and eventually surpass the acceptance level of traditional integration middleware (e.g., CORBA, Java RMI) [4]. This lack of capabilities is to a certain extent due to the way Web services interact with their environment that consists of peers, users, computing resources, just to name some. In fact a Web service only processes requests of users or peers without considering its internal execution state, or even questioning if it would be rewarded for processing these requests (e.g., will the Web service be privileged over other peers during selection?). In addition to this lack of capabilities, Web services are still considered by
Web services self-management by being flexible, stable, and autonomous. Flexibility means the capacity of a Web service to select the operations that accommodate the situation wherein it participates. Stability means the capacity of a Web service to resist unforeseen changes while maintaining operation and recovering to normal levels of operation after disturbances. Finally, autonomy means the capacity of a Web service to accept demands of participation in compositions, to reject these demands in case of unappealing rewards, or possibly to recommend other peers for participation in these compositions. Flexibility, stability, and autonomy requirements shed the light on a new way of looking into Web services. Indeed, it is all about making Web services assess their current capabilities, ongoing commitments, and surrounding environment prior they bind to any new composition and thus, expose appropriate behavior. We refer to these Web services as context-aware.

Making context-aware Web services expose appropriate behaviors in response to changes detected in the execution environment needs to be backed with appropriate means. We suggest using policies that permit to first, control the participation of Web services in composition scenarios and second, manage the life cycle of these scenarios. This life cycle spans over multiple steps including Web services discovery, selection, monitoring, etc. The variety and complexity of these steps yield insight into the different types of policies that need to defined. For example, the monitoring step calls for policies that make Web services react to delays in data submission.

The seamless combination of policies, Web services, and context is reported in Fig. 1. It shows how a Web service can be enriched through context with details on the environment. Such details concern users, computing resources, communication networks, other Web services, etc. This Web service "keeps an eye" on all the changes in the environment so that its dedicated context gets regularly updated with the previously cited details. Based on these details, a policy engine triggers policies and makes this Web service expose a certain behavior based on the outcomes of executing these policies. For example, a Web service could expose an acceptance behavior towards users’ needs satisfaction, if there are no security threats.

![Figure 1: Policy-driven, context-aware Web service](image)

In addition to context and policies, the management of Web services’ respective behaviors is affected by two extra elements. The user element identifies the personalization that Web services are subject to, and the resource element identifies the computing means upon which Web services operate. On the one hand, the dynamic nature of users’ expectations and requirements stresses the importance of including their preferences (e.g., preferred execution time) during Web services composition. On the other hand, because computing resources schedule the concurrent execution requests of Web services, these latter have to be constantly aware of these resources’ capacities and limitations. Upon resource identification and "bearing in mind" users’ preferences, a Web service checks with the resource if there is room for an additional execution. The resource assesses its ongoing and future commitments towards other Web services, and either accepts or rejects the execution request of this Web service. A rejection can be motivated by the constraints over a resource such as potential risk of overload.

While much of the work on Web services to date has focussed on low-level standards for publishing, discovering, and triggering Web services, we discuss and illustrate in this paper our proposed...
4-layer approach for developing context-based behavioral Web services using policies (Fig. 2). On top exists the policy layer, which relies on the information that context returns on the rest of layers namely user, Web service, and resource. Policies are triggered based on changes in users’ preferences, Web services’ states, and resources’ commitments. The rest of this paper is organized as follows. Section 2 defines the concepts of context and policy, presents the 4-layer approach that integrates policies into Web services composition and discusses how behavioral Web service get developed. The specification of policies of such Web services is presented in Section 3. In Section 4 the implementation of the prototype along with a running scenario are discussed. Related and future works are provided in Section 5 and Section 6, respectively. Finally, conclusions are drawn in Section 7.

2 The 4-layer approach

2.1 Brief definitions

The 4-layer approach that makes Web services more responsive to the environment in which they are situated is based on two main elements: context and policies.

- **Context** "... is not simply the state of a predefined environment with a fixed set of interaction resources. It is part of a process of interacting with an ever-changing environment composed of reconfigurable, migratory, distributed, and multiscale resources" [14]. Two scenarios illustrate the usages of context in Web services [26]: conditional engagement and self-management. For example, the conditional-engagement scenario shows that Web services do not always accept participating in composition scenarios. Prior to accepting, they check their execution state and question if they would be rewarded.

- **Policies** are defined as information which can be used to modify the behavior of a system [24]. Another definition, which goes beyond behavior modification, considers policies as external, dynamically modifiable rules and parameters that are used as input to a system [25]. This permits to the system to adjust to business and organizational decisions and changes in the execution environment. Six scenarios illustrate the usages of policies in Web services [25]: announcement, selection, compatibility, agreement, verification, and composition. For example, the announcement scenario shows how policies can be part of the WSDL description of a Web service. And the agreement scenario shows how users and providers of Web services could agree on the policies to use prior to execution.
2.2 Description of the four layers

In the following, we explain the role and contributions of each layer in the 4-layer proposed approach (Fig. 2).

The resource layer . It represents the computing means upon which Web services operate. The scheduling of resources because of Web services’ concurrent execution requests, happens when enough resources are not available to satisfy these requests all at once. To support and oversee this scheduling, a resource is associated with a set of arguments reported in Table 1 and listed as follows: label, previous periods of time/services, current period of time/services, next periods of time/services, previous locations/services, current location/services, next locations/services, and date. These arguments primarily manage a resource from time and location perspectives. The time perspective is about the periods of time that have featured/are featuring/will feature the execution of Web services over a resource. The location perspective is about the locations that have simultaneously featured/are simultaneously featuring/will simultaneously feature both the presence of users in certain locations and the execution of Web services over a resource.

<table>
<thead>
<tr>
<th>Argument &amp; Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>corresponds to the identifier of the resource.</td>
</tr>
<tr>
<td>Previous periods of time/services</td>
<td>keeps track of the periods of time, as indicated by the user, that have featured the execution of services with regard to each composite service (null if there are no predecessor periods of time). The effective periods of time of the execution of services are also reported in this parameter.</td>
</tr>
<tr>
<td>Current period of time/services</td>
<td>indicates the current time, as indicated by the user, that should now feature the execution of services with regard to each composite service.</td>
</tr>
<tr>
<td>Next periods of time/services</td>
<td>keeps track of all the periods of time, as indicated by the user, that will feature the execution of services with regard to each composite service (null if there are no next periods of time).</td>
</tr>
<tr>
<td>Previous locations/services</td>
<td>keeps track of the locations, as indicated by the user, that have featured the execution of services with regard to each composite service (null if there are no predecessor periods of time). The effective locations of the execution of services are also reported in this parameter.</td>
</tr>
<tr>
<td>Current location/services</td>
<td>indicates the current location, as indicated by the user, that should now feature the execution of services with regard to each composite service.</td>
</tr>
<tr>
<td>Next locations/services</td>
<td>keeps track of all the locations, as indicated by the user, that will feature the execution of services with regard to each composite service (null if there are no next periods of time).</td>
</tr>
<tr>
<td>Date</td>
<td>identifies the time of updating the parameters above.</td>
</tr>
</tbody>
</table>

The Web service layer . It represents the Web services that are posted on various registries like UDDI so that users can identify them according to their needs and execute them later. This execution is in compliance with resource availability as shown in the resource layer. Like a resource, a Web service is associated with a set of arguments reported in Table 2 and listed as follows: label, state per participation, previous services per participation, next services per participation, regular actions, start time per participation (requested and effective), location per participation (requested and effective), reasons of failure per participation, corrective actions per participation, and date. These arguments describe a Web service according to execution chronologies in terms of pre- and post-Web services, time and location parameters that will trigger the execution of the Web service, and corrective measures in case of execution failures.

In Tables 1 and 2, ”each composite service” and ”per composition” expressions indicate the capacity of a Web service to simultaneously participate in several compositions. This highlights the Web services instantiation principle that was introduced in [27]. A direct benefit
Table 2: Arguments associated with Web service

<table>
<thead>
<tr>
<th>Argument &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label: corresponds to the identifier of the Web service.</td>
</tr>
<tr>
<td>State per participation: informs about the current state of the service with regard to each composite service in which the service takes part. State can be of types in-progress, suspended, aborted, or terminated.</td>
</tr>
<tr>
<td>Previous services per participation: indicates whether there are services before the service with regard to each composite service (null if there are no predecessors).</td>
</tr>
<tr>
<td>Next services per participation: indicates whether there are services after the service with regard to each composite service (null if there are no successors).</td>
</tr>
<tr>
<td>Regular actions: illustrates the actions that the service normally performs.</td>
</tr>
<tr>
<td>Start time per participation (requested and effective): informs when the execution of the service should start as requested by the user (i.e., user-related), and has effectively started with regard to each composite service (i.e., execution-related).</td>
</tr>
<tr>
<td>Location per participation (requested and effective): informs where the execution of the service should happen as requested by the user (i.e., user-related) and has effectively happened (i.e., execution-related) with regard to each composite service.</td>
</tr>
<tr>
<td>Reasons of failure per participation: informs about the reasons that are behind the failure of the execution of the service with regard to each composite service.</td>
</tr>
<tr>
<td>Corrective actions per participation: illustrates the actions that the service has to perform in case its execution fails. The type of actions depends on the reasons of failure.</td>
</tr>
<tr>
<td>Date: identifies the time of updating the parameters above.</td>
</tr>
</tbody>
</table>

of this principle is the temporal decomposition of a Web service into three categories: participation in compositions already happened (past), participation in compositions currently happening (present), and participation in compositions to potentially happen (future).

The user layer. It represents users who look for Web services with respect to their needs that vary from travel planning to book ordering. Users may require some assistance in locating, selecting, and composing the appropriate Web service. This could be done using for example software agents [11], but this is outside this paper’s scope. Like a resource and Web service, a user is associated with a set of arguments reported in Table 3 and listed as follows: label, previous locations/services, current location/services, next locations/services, previous periods of time/services, current period of time/services, next periods of time/services, and date. These arguments describe a user from time and location perspectives. The time perspective is about the periods of time that have featured/are featuring/will feature the execution of Web services as per user’s request. The location perspective is about the locations that have featured/are featuring/will feature the execution of Web services due to the location of user.

Table 3: Arguments associated with user

<table>
<thead>
<tr>
<th>Argument &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label: corresponds to the identifier of the user.</td>
</tr>
<tr>
<td>Previous locations/services: keeps track of all the locations, as indicated by the user, that have featured in the past the execution of services (null if there are no predecessor locations).</td>
</tr>
<tr>
<td>Current location/services: indicates the current location, as indicated by the user, that should feature now the execution of services.</td>
</tr>
<tr>
<td>Next locations/services: indicates all the locations, as indicated by the user, that will feature the execution of services (null if there are no next locations).</td>
</tr>
<tr>
<td>Previous periods of time/services: keeps track of all the periods of time, as indicated by the user, that have featured the execution of services (null if there are no predecessor periods of time).</td>
</tr>
<tr>
<td>Current period of time/services: indicates the current time, as indicated by the user, that should feature now the execution of services.</td>
</tr>
<tr>
<td>Next periods of time/services: keeps track of all the periods of time, as indicated by the user, that will feature the execution of services (null if there are no next periods of time).</td>
</tr>
<tr>
<td>Date: identifies the time of updating the parameters above.</td>
</tr>
</tbody>
</table>
The policy layer. It represents the policies that define how a Web service needs to react based on the progress of a composition. This progress is subject to the interactions a Web service has with users and resources. Interactions between users and Web services concern handling and validating\textsuperscript{1} users’ preferences. Interactions between Web services and resources concern the mechanisms of supporting the execution of personalized Web services subject to the constraints on these resources.

2.3 Definition of Web services’ behaviors

In the 4\textsuperscript{-}layer approach, policies make Web services expose certain behaviors based on the execution progress of a composition. An example of composition is provided in Section 4.2. Four types of behavior are identified: permission, restriction, dispensation, and obligation\textsuperscript{2}. Providers of Web services are in charge of specifying the policies per type of behavior (Section 3). In [28], Marjanovic considers some of these behaviors as part of the normative perspective that shows the parties involved in Web services provisioning. In the following, we describe each behavior type and then, discuss how a Web service manages in binding to a behavior.

1. Permission: a Web service accepts the invitation of participation in a composite Web service upon validation of its current engagements in other composite Web services.

2. Restriction: a Web service cannot be part of a composite Web service due to non-compliance of some component Web services in this composite Web service with this Web service’s non-functional requirements.

3. Dispensation means that a Web service breaks either a permission or a restriction of participation in a composition. In the former case, the Web service does not participate despite the acceptance that is granted. In the latter case, the Web service does participate despite the restrictions that are detected. When it comes to permission, an example of dispensation could be the unexpected breakdown of a resource upon which the performance of the Web service is scheduled. And when it comes to restriction, an example of dispensation could be the priority level of a user, which requires an immediate handling of his needs by the Web service.

4. Obligation: this is a strong permission for a Web service to participate in a composite Web service in spite of negative permission of participation or existence of restrictions from participation. Obligations override dispensations related to permission or restriction.

Fig. 3 illustrates the way the different behaviors of a Web service are related to each other based on the execution outcome of policies. In this figure, behaviors are represented with ellipses and links between these ellipses are labeled with yes or no. In the same figure, dispensation\textsubscript{P} and dispensation\textsubscript{R} stand for dispensation related to permission and related to restriction, respectively. In addition, participation\textsubscript{(+)} and participation\textsubscript{(-)} stand for positive and negative participation in composition, respectively.

The chronology of operations in Fig. 3 happens as follows. Initially, a Web service is invited to take part in a composite Web service. The decision of either accepting or rejecting the invitation is made on the basis of the outcome of executing the permission-related behavior policy. If a permission is granted, the next step is to check if there is any dispensation that could invalidate this permission. If the outcome of executing the dispensation\textsubscript{P}-related behavior policy is positive, this will result in canceling the positive permission of participation. Otherwise, the next step

\textsuperscript{1} Users could have conflicting preferences like being committed to be in separate locations but at the same time.

\textsuperscript{2} The use of similar attributes can be found in Intelligent Artificial systems specified using the deontic logic. For instance, a deontic logic uses O(A) to mean it is obligatory that action A and P(A) to mean it is permitted (or permissible) that action A.
is to check if there is any restriction that could prevent this Web service from participating in this composite Web service. If the outcome of executing the restriction-related behavior policy is negative, the initial positive permission of participation is finally confirmed. Otherwise and like dispensations for permissions, dispensations to cancel restrictions are checked as reported earlier using the dispensation $R$-related behavior policy.

Fig. 3 shows the different paths that result in the participation of a Web service in a composition. For instance, a path combines a negative permission (no) and then a positive obligation (yes). Another path combines a positive permission (yes), then a negative dispensation of type permission (no), and finally a negative restriction (no). The way all the different paths could be generated out of this figure can be presented using Table 4 where each case corresponds to a composite Web service $i$ ($CWS_i$). For instance, a negative authorization is assigned to $CWS_1$ due to the double negative permission and obligation it gets. A positive authorization is accorded to $CWS_2$ due to the positive obligation it gets and this regardless of the the negative permission. Finally, a positive authorization is accorded to $CWS_3$ which gets a positive permission without any dispensation and restriction.

### Table 4: Some scenarios showing final authorizations of participation in compositions

<table>
<thead>
<tr>
<th>Row #</th>
<th>Cases</th>
<th>Permission</th>
<th>Dispensation</th>
<th>Restriction</th>
<th>Dispensation</th>
<th>Obligation</th>
<th>Final authorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Case1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Case2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>Case3</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>

### 2.4 Representation of Web services’ behaviors

We are investigating the design/representation of the four types of behaviors that a Web service can bind to using the well-known state design pattern [19]. Design patterns provide proven solutions
to recurring software design problems. They improve software understandability and reusability and are gathered and documented in patterns catalogs\(^3\). The state pattern lets an object take on a state out of several following changes in its internal behavior. It works by encapsulating the individual states along with the corresponding state changing logic into independent classes, thus removing/minimizing conditional logic. For Web services, changes in states reflect the result of executing policies. Fig. 4 illustrates how the behavior of type permission is modeled in compliance with the state design pattern. The diagram shows that Permission class is abstract and BehaviorManager object contains a Permission object (objects are obtained after instantiation at run time). The Permission class is a placeholder. Each subclass defines the behavior that is appropriate to a certain state.

![Diagram of state design pattern]

Figure 4: Modeling behaviors (permission type) using the state design pattern

For illustration purposes, let us adopt the negative permission (no) and then the positive obligation (yes) path (Fig. 3, Table 4, row # 2). The permission behavior can bind to either positive (yes) or negative (no) state. After policy execution (Section 3.2) using the trigger method, the permission of participation is rejected. Not shown in this figure (Fig. 4), the obligation policy that gets triggered as well following this rejection. When the BehaviorManager object receives a request to trigger the appropriate policies, it delegates the request to its state object. The types of policies triggered depends on the state that the permission is in at that time. Rather than writing complex conditional statements, the individual state objects define how the methods are to behave for that state.

Similarly, each of the three remaining behaviors (i.e., dispensation, restriction, and obligation) is modeled using this pattern. The main difference concerns the obligation behavior, which will finally, either confirm the invitation of the Web service to participate in the composition or deny to the invitation. The latter can be set by sending a refusal notification to the requester Web service.

### 2.5 Policies during interactions

In a Web services composition, interactions are classified as either vertical (between composite Web service and its component Web services) or horizontal (between component Web services in the same composite Web service). Through interaction, the initiator aims at making the recipient act upon the conveyed messages by taking actions, which affects this recipient’s behavior. In the following we discuss how policies are side-effects of these interactions.

In vertical interactions, a centralized orchestration of Web services is implemented. This type of orchestration is adopted in systems like eFlow [12] and CMI [32]. Here, a composite Web service has the authority to carry out the following actions over a component Web service (Fig. 5):

---

\(^3\)www.hillside.net/patterns/onlinepatterncatalog.htm.
- "invite" action permits the composite Web service to request the participation of the Web service in the composition this composite Web service leads;

- "ping" action permits the composite Web service to check the liveness of the Web service that accepted its invitation of participation; there is no guarantee that a Web service in a composition is still available at time of request;

- "trigger" action permits the composite Web service to initiate the execution of the Web service that accepted its invitation of participation;

- "audit" action permits the composite Web service to monitor the performance to the Web service for assessment purposes; and,

- "retract" permits the composite Web service to withdraw the Web service in case of poor performance. This results in starting the search for another Web service to append into the composition this composite Web service leads.

When a composite Web service decides to perform "invite", "ping", "trigger", "audit", or "retract" actions, the concerned component Web service checks its context prior to triggering policies and responding to the composite Web service. In this paper, we focus on policies related to composition participation through "invite" action (Fig. 3). As part of our future work other types of policies are planned. For example to trigger a Web service for execution, this should happen in compliance with the policies that first, grant the positive participation of this Web service and second, verify the acceptable triggering times.

In horizontal interactions, a peer-to-peer orchestration of Web services is implemented. This type of orchestration is adopted in systems like Self-Serv [6] and PCAP [33]. Here, a component Web service has the authority to carry out the following actions over another peer in the same composition (Fig. 6):

- "ping" action permits the Web service to check the liveness of a peer; there is no guarantee that a Web service in a composition is still available at time of request;

- and, "trigger" action permits the Web service to initiate the execution of a peer.

---

*Service level agreements motivate auditing Web services [23].*
Similar to the role of policies in vertical interactions, actions in horizontal interactions result in checking context and triggering policies. For example, monitoring a Web service execution should happen in compliance with privacy policies.

3 Specification of policies

We specify the policies that restrict the acceptable behavior of a Web service using the Web Services Policy Language (WSPL). As per Fig. 3, four types of policies are needed. Our selection of WSPL as well as comparison between WSPL and WS-Policy is backed by [3].

3.1 Overview of WSPL

The selection of a policy specification language is guided by some requirements that need to be satisfied [34]: 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 WSPL is used. Anderson mentions that a Web service has various aspects and features that can be controlled or described using policy rules [2]. Some of these aspects include authentication, quality-of-service, privacy, and reliable messaging. Anderson suggests WSPL to express policies and achieve the interoperability of Web services. The following illustrates an example of a WSPL policy in an abstract form for readability purposes. The semantics of this syntax is that if either rule is satisfied, then the policy is satisfied. WSPL allows policy negotiation by supporting the merging of policies from two sources. In addition, policies can be based on policy parameter comparisons other than simple equality matching. The syntax of WSPL is strictly based on the OASIS eXtensible Access Control Markup Language (XACML) standard.

```xml
Policy (Aspect="Quality of Protection") {
    Rule{ Signature-Algorithm = "RSA-SHA1", Key-Length >= 2048}
    Rule{ Signature-Algorithm = "RSA-SHA1", Key-Length >= 1024, Source-Domain = "EXAMPLE.COM"}
}
```

3.2 Permission policy

Permission consists of authorizing a Web service to join a composite Web service. The following illustrates a permission policy in WSPL. It states that a Web service participates in a composition subject to evaluating <Condition> to true. This latter refers to some arguments like the current number of active participations of the Web service in compositions and the next possibility of participation of the Web service in additional compositions. Both arguments keep track of the state of a Web service (Table 2). Arguments are known as vocabulary items in WSPL. In the presented policy, <TrueConclusion> shows the permission of participation, whereas <FalseConclusion> shows the contrary.

```xml
Policy (Aspect="Permission") {
    <Rule xmlns="urn:oasis:names:tc:xacml:3.0:generalization:policy:schema:wd:01" RuleId="PermissionWS">
        <Condition>
            <Apply FunctionId="and">
                <Apply FunctionId="integer-less-than" DataType="boolean">
                    <SubjectAttributeDesignator AttributeId="CurrentNumberOfParticipations" DataType="integer"/>
                    <SubjectAttributeDesignator AttributeId="MaximumNumberOfParticipations" DataType="integer"/>
                </Apply>
                <SubjectAttributeDesignator AttributeId="Availability" DataType="boolean"/>
            </Apply>
        </Condition>
    </Rule>
}
```

For the sake of simplicity, we do not list in the aforementioned policy the full URNs for the XACML DataTypes and FunctionIds, and use rather simple names for the vocabulary items (e.g., CurrentNumberOfParticipations) and simple names for function identifiers (e.g., integer-less-than). The vocabulary items could also be elements in a schema instance that is passed to the policy evaluator, in which case "AttributeSelector" with an XPath expression would be used rather than <...AttributeDesignator>.

3.3 Restriction policy

Restriction aims at preventing a Web service from participating in a composite Web service. Restriction neither backs nor follows a negative permission of participation. However, it implements a positive permission of participation that is reinforced with a non dispensation from this participation (Fig. 3). Restrictions could be related to the QoS (e.g., time of response, performance, throughput) [30] of the component Web services with whom a Web service will interact. A Web service’s provider can only have interest in the Web services that have a “good” QoS record. The following illustrates a restriction policy in WSPL. It states that a Web service can be restricted from participation subject to evaluating <Condition> to true. This latter checks that a positive permission of participation exists, a non dispensation from participation exists too, and last but not least the assessment level of the QoS of the Web services is low. In this policy, QoSAssessment is an integer value that is the result of evaluating the QoS of a Web service, and QoSThreshold is the minimum QoS assessment value that is acceptable for a composition.

```
Policy (Aspect="Restriction") { 
  <Rule xmlns="urn:oasis:names:tc:xacml:3.0:generalization:policy:schema:wd:01" RuleId="RestrictionWS">
    <Condition>
      <Apply FunctionId="and">
        <SubjectAttributeDesignator AttributeId="YesPermission" DataType="boolean"/>
        <Apply FunctionId="integer-great-than-or-equal">
          <SubjectAttributeDesignator AttributeId="QoSAssessment" DataType="integer"/>
          <SubjectAttributeDesignator AttributeId="QoSThreshold" DataType="integer"/>
        </Apply>
        <SubjectAttributeDesignator AttributeId="NoDispensationP" DataType="boolean"/>
      </Apply>
    </Condition>
    <Conclusions>
      <TrueConclusion Restriction = "Permission(-)"/>
      <FalseConclusion Restriction = "Permission(+)"/>
    </Conclusions>
  </Rule>
}
```

3.4 Dispensation policy

Dispensation allows a Web service to break policies related to either permission or restriction. To this end, permission-driven dispensation and restriction-driven dispensation are identified. For illustration reasons, we only describe the first type. Although a Web service has been granted a permission to be part of a composite Web service (Fig. 3), a dispensation from participation can override this permission. One of the criteria that backs the dispensation is the invocation period of the composite Web service to the Web service. If this period falls into the Web service’s peak-time period of receiving invocation requests, the Web service can cancel the permission of participation. This might affect the QoS that this Web service guarantees to other composite Web services. The following illustrates a permission-driven dispensation policy in WSPL. It states that a Web service cancels a permission of participation subject to evaluating <Condition> to true.
This latter checks that such a permission exists and the invocation-request time does not fall into the peak time period of the Web service.

```
Policy (Aspect="DispensationPermission") {
  <Rule xmlns="urn:oasis:names:tc:xacml:3.0:generalization:policy:schema:wd:01"
    RuleId="DispensationPermissionWS">
    <Condition>
      <Apply FunctionId="and">
        <SubjectAttributeDesignator AttributeId="YesPermission" DataType="boolean"/>
        <SubjectAttributeDesignator AttributeId="PeakTime" DataType="integer"/>
        <SubjectAttributeDesignator AttributeId="RequestTime" DataType="integer"/>
      </Apply>
    </Condition>
    <Conclusions>
      <TrueConclusion DispensationPermission = "Permission(-)"/>
      <FalseConclusion DispensationPermission = "Permission(+)"/>
    </Conclusions>
  </Rule>
}
```

### 3.5 Obligation policy

Obligation guarantees that a Web service will definitely be part of a composite Web service, despite a negative permission of participation, or a dispensation from participation, or a dispensation from restrictions (Fig. 3). Obligations are aimed for the situations that call for an immediate handling of users’ needs. In addition, obligations might result in dropping or suspending some of the ongoing compositions due to the maximum number of participations of a Web service in multiple compositions at a time [27]. Because of this maximum number of participation, a Web service might have to suspend or drop some of its ongoing participations due to the obligation of joining another composition. Compensation solutions are to be developed for the compositions that are affected by suspension or dropping decisions of participation. For example, a substitute Web service needs to be found if a Web service decides to cancel its participation in an ongoing participation. Furthermore, a composition needs to be resumed according to a certain time frame if a Web service temporarily suspends its participation. The following illustrates an obligation policy in WSPL. It states that a Web service takes part in a composite Web service subject to evaluating `<Condition>` to true. This latter checks that whether a no-permission of participation exists, a dispensation from participation exists, or a no-dispensation from restriction exists too.

```
Policy (Aspect="Obligation") {
  <Rule xmlns="urn:oasis:names:tc:xacml:3.0:generalization:policy:schema:wd:01"
    RuleId="ObligationWS">
    <Condition>
      <Apply FunctionId="and">
        <Apply FunctionId="xor">
          <SubjectAttributeDesignator AttributeId="NoPermission" DataType="boolean"/>
          <SubjectAttributeDesignator AttributeId="YesDispensationP" DataType="boolean"/>
          <SubjectAttributeDesignator AttributeId="NoDispensationR" DataType="boolean"/>
        </Apply>
        <Apply FunctionId="equal">
          <SubjectAttributeDesignator AttributeId="UserNeedLevel" DataType="string"/>
          <AttributeValue DataType="integer"/>
          "high"
        </Apply>
      </Apply>
    </Condition>
    <Conclusions>
      <TrueConclusion Obligation = "Permission(+)"/>
      <FalseConclusion Obligation = "Permission(-)"/>
    </Conclusions>
  </Rule>
}
```
4 Implementation

We discuss the work that was conducted for implementing the 4-layer approach for Web services composition prior to discussing how this approach applies to a running scenario.

4.1 Prototype architecture

Fig. 7 illustrates the architecture of the prototype in which numbers illustrate the chronology of operations. For compatibility purposes, we used Sun Microsystems’s tools namely J2EE 1.4 to develop Web services and XACML Open Source to develop policies defined in WSPL. Eclipse 3.1 is used as a integrated development environment. The prototype consists of two registries (UDDI and policies) and four managers. The composition manager supports developers during the specifica-

The component manager is responsible for identifying the Web services that satisfy users’ needs after UDDI-registry consultation. The list of Web services to identify is driven by users’ needs and provided by the composition manager. Afterwards, the identified Web services are submitted to the policy manager, which will make Web services bind to appropriate behaviors (either permission, dispensation, restriction, or obligation) according to the various contextual details it receives from the context manager. Examples of details could be related to resource availabilities and are reported in Tables 1, 2, and 3. We recall that the final outcome of executing policies is to either accept or reject to participate in a composition. Last but not least the context manager is built on top of a Context Toolkit (www.cs.cmu.edu/~anind/context.html), a package that supports the development of context-aware applications.

4.2 Running scenario

Our running scenario concerns Amin who is visiting Melissa in Oslo, Norway. Amin and Melissa agree to meet in a coffee shop in downtown. Amin has two options to reach the meeting place: by taxi or by bus. A rough specification of this scenario is illustrated with Fig. 8. This specification could be done in BPEL for example, without any changes in the various policies that were previously defined.

At his hotel, Amin browses some Web sites about transportation in Oslo. A site has Itinerary WS that proposes routes between two specific places like Amin’s hotel and the coffee shop. The proposed routes are subject to weather forecasts: cold weather results in recommending taxis, otherwise public transportation like tramways and buses are recommended. Parallel to consulting
with Weather WS, Itinerary WS requests details about the origin and destination places using Location WS.

In case Weather WS forecasts bad weather, a taxi booking is made using Taxi WS upon Amin’s approval. Otherwise, i.e., pleasant day, Amin uses public transportation. The locations of Amin’s hotel and coffee shop are submitted to Bus Schedule WS, which returns for example the bus numbers that Amin has to ride. Potential traffic jams force Bus Schedule WS to regularly interact with Traffic WS that monitors the status of the traffic network. This status is fed into Bus Schedule WS so that adjustments in bus numbers can occur.

Fig. 9 shows some snapshots of the prototype we developed for bus-scheduling management. The top part of the figure shows the different options that are suggested to a developer. For instance, upon selection of Weather WS, he could either invoke it or specify its policies or context. In the bottom part of the figure, the snapshot illustrates the permission to participate in a composition, which is given to Weather WS after executing the appropriate policies.

5 Related work

Because this paper’s discussion is about behavioral Web services using policies, related work on context-aware Web services is excluded. For further details on this topic readers can consult for instance [10, 26, 29]. The works reported in this section adopt policies as a mechanism for restricting the way systems should respond to events.

Policies have been applied in multiple domains like telecommunication-devices control features [31] and conversation regulation between intelligent components [21]. In the field of Web services, policies can specify different facets of the behavior of a Web service so that this Web service’s capabilities can be aligned to users’ requirements and resources’ constraints. In [25], Maamar et al. list six different scenarios regarding the potential uses of policies in Web services namely announcement, selection, compatibility, agreement, verification, and composition. For example, the announcement scenario shows how offered and required policies can be part of the WSDL description of a Web service. Moreover Maamar et al. discuss how the lack of a standard framework for enhancing Web services through policies is currently balanced with two approaches namely semantic Web and Boolean combinations of policy predicates.

In [22], Kagal et al. claim that policies should be part of the representation of Web services, and in particular semantic Web services. Policies provide the specification of who can use a Web service under what conditions, how information should be provided to a Web service, and how this provided information will be used later. When it comes to Desai et al., they recognize the importance of concern separation from a software engineering perspective [16]. Desai et al. split a business process into two parts: protocol and policy. The protocol part is a modular, public specification of an interaction among different roles that collaborate towards achieving a desired goal. The policy part is a private description of a participant’s business logic that controls how this participant takes part in a protocol.

In [5], Baresi et al. propose a policy-based approach to monitor Web services’ functional
(e.g., constraints on exchanged data) and non-functional (e.g., security, reliability) requirements. They report on the different types of policies that can be defined like service policies, server policies, supported policies, and requested policies. Another work related to using policies to manage Web services composition is reported in [1]. Here, Ae Chun et al. propose a formal model that includes service and user policy. Service policy are imposed by Web services' providers. It should be noted that any selected Web service for a given composition scenario can guarantee that its policies are first, compatible with user policies and second, satisfy syntactic, semantic, and policy-level compatibility. In Ae Chun et al.'s work, user policies refer to users' constraints and preferences to select particular Web services.

Policies have also been used to access and use resources in other fields. In grid computing, Du-mitrescu et al. describe two strategies that a virtual organization must deploy to ensure grid-wide resource policy enforcement in a decentralized manner [18]. In the semantic Web, Gavriloaie et al. present an approach to manage access to sensitive resources using PeerTrust, a language for expressing access control policies [20]. PeerTrust is based on guarded distributed logic programs and has been demonstrated for automated trust negotiation.

6 Future work - policy protection

In the previous sections, we discussed how a Web service binds to a particular behavior according to the current state of its surrounding environment. To guarantee that this binding happens as
expected, we aim at protecting policies from unwanted alterations as well as misuses by unauthorized parties. Policies are resources that can be subject to attacks like any other resource offered to external parties. For instance, a Web service could participate in a composition thought it did not obtain the permission. The appropriate policy did not get executed due to unpermitted changes in this policy’s content. Although the security of Web services is cornerstone in any composition, we refer readers to our previous work [27].

Our strategy to protect behavior policies of Web services is built upon another set of dedicated security policies. According to Wang et al., the term security policy has to come to mean different things to different communities [35]. For example, an access-control policy defines who has access to what and under what circumstances. Moreover, Clemente et al. mention that in the Web information systems security field, a security policy can be defined as a set of rules and practices that describe the way an organization manages, protects, and distributes sensitive information at several levels [13].

Considering two types of policies, those in charge of the behavior of Web services and those in charge of securing these policies, is backed by the work of Dimitrakos et al. [17]. They claim that posting policy statements as resources on the Web makes them vulnerable to attacks and thus, the security of the whole site is compromised. Dimitrakos et al.’s proposal consists of applying policies to policies. One can then use “hidden” policies to control access to “public” policies. “Hidden” and “public” policies of Dimitrakos et al. are mapped onto our security policies and behavior policies, respectively. The same argument for using “hidden” and “public” policies is cited by Anderson [3], who suggests “internal” and “public” policies. In the following we discuss first, the risks that put in danger the consistency of behavior policies of Web services and second, the countermeasures in terms of security policies that enable minimizing these risks.

**Threats on policies.** We decompose the threats on a policy into two types: content alteration and prohibited use. On the one hand, content alteration means setting values of some arguments like `QoSThreshold` and `MaximumNumberOfParticipations` of a policy to unacceptable values. This setting could for example delay the triggering of a policy or authorize its triggering despite restriction. On the other hand, prohibited use means allowing a third party to exploit a policy, although this is not authorized due to access restrictions.

**Specification of security policies.** For compatibility purposes with the behavior policies of Web services (Section 3), we plan to adopt WSPL to specify security policies. Our primary motive for these policies is to frame the way behavior policies of Web services are managed in terms of consultation and update. This allows to know for example who did use a policy, when was a policy triggered, what was the execution outcome of a policy was, etc. We plan to comply with the Web service’s Quality of Protection (QoP) approach of Anderson [2].

### 7 Conclusion

In this paper, we presented a 4-layer approach to manage behavioral Web services engaged in compositions. Current standards do not suggest how to make Web services more responsive to the environment, and existing composition approaches assume the automatic participation of Web services in compositions but neglect the cases where these Web services could be overloaded and would therefore prefer to either delay or reject their participation. Our approach comprised four layers referred to as resource, Web service, user, and policy, respectively. Interesting to highlight the role of the policy layer, which, for instance, enabled the separation of concerns between the requirements of the composition level (i.e., composite Web service) and the requirements of the component level (i.e., Web service). This separation of concerns was backed by the information that context provides over preferences of users, execution states of Web services, and commitments of resources towards Web services. Furthermore, we promoted the importance of Web services
adaptability by allowing them to expose distinct behaviors according to the situation wherein they evolve. This behavior was organized along four interconnected perspectives referred to as permission, obligation, restriction, and dispensation. Our prototype exposes a set of characteristics summarized in the following:

- Support for contextual information including gathering and management using a context toolkit.
- Clear separation between the Web service and the implementation of its behavior by implementing a manager per component (policies, context, composition).
- And, use of widely adopted technologies and tools (XACML, context toolkit, etc.) to allow an easy extension of the prototype.

In addition to completing our strategy for protecting policies, an additional research avenue concerns conflicts between policies. This calls for solving strategies that would let policies override each other as well as using argumentation between conflicting policies [8].

Acknowledgments
The authors would like to thank Anne H. Anderson from Sun Microsystems Laboratories in Burlington, MA for the fruitful discussions on Web services policy standards. In addition, they would like to thank the anonymous reviewers for their valuable comments and suggestions of changes.

References


