Security Conscious Web Service Composition[1]

Barbara Carminati[1], Elena Ferrari[1], Patrick C. K. Hung[2]
[1]*University of Insubria, Italy*
[2]*University of Ontario Institute of Technology (UOIT), Canada*
{barbara.carminati, elena.ferrari}@uninsubria.it; patrick.hung@uoit.ca

Abstract

A Web service is a software system designed to support interoperable application-to-application interactions over the Internet. Web services are based on a set of XML standards, such as Web Services Description Language (WSDL), Simple Object Access Protocol (SOAP) and Universal Description, Discovery and Integration (UDDI). Recently, there has been a growing interest in Web service composition, and some languages (e.g., WSBPEL, BPML) for modeling the composition have been proposed. In this paper, we focus on security constraints of Web service composition, which have not been deeply investigated so far. We propose a method for modeling security constraints and a brokered architecture to build composite Web services according to the specified security constraints.

Keywords: *Web services, workflow, security constraints, WSBPEL*

1. Introduction

A Web service is a software system designed to support interoperable application-to-application interactions over the Internet. Web services rely on a set of XML standards such as Universal Description, Discovery and Integration (UDDI) [14], Web Services Description Language (WSDL) [15], and Simple Object Access Protocol (SOAP) [16]. A business process contains a set of activities that represent both business tasks and interactions. One of the major goals of Web services is to make easier their composition to form more complex services. To this purpose, many emerging languages (e.g., BPEL4WS [5], WSBPEL [13] and BPML [2]) have been proposed for coordinating Web services into a workflow. A workflow is a computer supported business process. The information processed in a workflow may be highly valued and thus it is important to protect this information against security threats. The prolific use of workflow management systems for critical and strategic applications gives rise to a major concern regarding the threats against confidentiality, integrity, privacy, anonymity, and availability. Additionally, the BPEL4WS specification recommends that business process implementations use WS-Security [6] to ensure messages have not been modified or forged while in transit or while residing at destinations. In this paper, we consider an aspect of Web service composition that has not been so far deeply investigated, despite its importance, that is, security. The idea is that both Web service requestors and providers may have security requirements and properties that must be taken into account when composing Web services. We refer to Web service composition driven by security requirements as *security conscious composition*. For instance, a Web service provider may not want to accept requests issued by a specific IP address, or it may want to put some additional security constraints on the composition. Such constraints must be carefully considered when composing Web services. In this paper, we first propose a way to model such security constraints, which is compliant with existing standards. Then, we present a brokered architecture to compose Web services according to the specified security constraints.

The remainder of this paper is organized as follows. Next section discusses related work. Section 3 illustrates our strategy to model security constraints and capabilities. Section 4 describes the architecture on support of security conscious web service composition, whereas Section 5 illustrates the Security Matchmaker, which is the core of the proposed architecture. Section 6 describes a prototype implementation of the proposed framework. Finally, Section 7 concludes the paper.

2. Literature Review

In the past few years, business process or workflow proposals relevant to Web services are proliferating in the business and academic world. Most of the proposals are XML-based languages to specify Web services interactions and compositions. All of the proposed XML languages are based on WSDL service descriptions with extension elements. For example, the Business Process Execution Language for Web Services (BPEL4WS) is a

[1] The work reported in this paper has been partially supported by the discovery grant (NSERC PIN: 290666) from the Natural Science and Engineering Research Council (NSERC) of Canada under the project: "M-services computing security and privacy enforcement model."

formal specification of business processes and interaction protocols. The OASIS WSBPEL Technical Committee is now established to continue working on the BPEL4WS 1.1 specification within the OASIS Consortium [13]. WSBPEL defines a model and a grammar for describing the behavior of a business process based on interactions between the process and its Web service interfaces. In short, a WSBPEL business process definition can be thought of as a template for creating business process instances. Each of the activities in a flow model must be executed by an appropriate Web service. In this scenario, the role of service locators is to assign an appropriate Web services for each activity. This assignment process is called matchmaking. Besides exploiting the UDDI registers, the matchmaking process can be performed also by means of semantic Web service descriptions. In this context, DAML-S [1] provides capability to semantically annotate Web services based on an ontology that provides classes and properties to describe content and capabilities of the Web services. Another relevant effort carried on in this field is the one proposed in [9], where authors extend OWL-S, the new emerging standard for semantic Web service description, by proposing ontology for annotating input and output parameters of a Web service with respect to their security characteristics (e.g., encryption and digital signature requirements). In [9] they also consider privacy and authorization policies expressed by means of the REI language [8]. A basic difference between the approach reported in [9] and the one proposed in this paper is that we exploit a syntactic approach to model security requirements of a Web service (i.e., the WSDL document), whereas in [9] they use a semantic annotation-based approach. A further relevant difference is that in [9] the authors only consider the enforcement of security constraints of a single Web service requester. By contrast, in the proposed approach we consider the security requirements of both Web service requestors and Web services taking part in the composition.

Other related work is those exploiting AI planning techniques for Web service composition. Among them, we recall the work by McIlraith et al. [10] that extends the logic programming language Golog for automatic composition of Web services, the one by Medjahed [11], which proposes a technique for generating composite Web services from high-level declarative descriptions. A framework for composing Web services, based on the use of Mealy machines has also been proposed by Bultan et al. [3]. However, such frameworks do not address security issues such as access control, which is the focus of our work. There are also XML languages proposed for describing security assertions. These XML languages restrict access to Web services to authorized parties only, and protect the integrity and confidentiality of messages exchanged in a loosely coupled execution environment. Specifically there is a well-known format for XML-based security tokens, that is, the Security Assertions Markup Language (SAML), which is used to define authentication and authorization decisions in Web services [12]. Web services providers submit SAML tokens to security servers for making security decisions. WS-Security describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality and single message authentication [6]. Based on WS-Security, WS-Policy provides a grammar for expressing Web services security policies [7]. The WS-Policy includes a set of security policy assertions to support the WS-Security specification defined in WS-Security Policy [7].

3. Security capabilities and constraints

The starting point to model any security information related to Web services is defining a reference vocabulary. We define the Security Vocabulary/Ontology by using the Web Ontology Language (OWL) [21]. OWL ontology includes descriptions of classes, properties, and their instances, as well as formal semantics for deriving logical consequences in entailments. Figure 1 shows a simplified OWL Ontology that describes a security vocabulary and related Web services standards.

```
<owl:Ontology rdf:about="#securityVocabulary">
  <owl:versionInfo>v 1.00 2005/06/15 23:59:59</owl:versionInfo>
  <rdfs:comment>Security Vocabulary</rdfs:comment>
  ...
  <owl:Class rdf:ID="#privacyAccessControl">
    <owl:unionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#P3P"/>
      <owl:Class rdf:about="#EPAL"/>
      <owl:Class rdf:about="#XACML"/>
    </owl:unionOf>
  </owl:Class>
  ...
  <owl:Class rdf:ID="#authentication">
    <owl:unionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#WS-Security"/>
      <owl:Class rdf:about="#SAML"/>
      <owl:Class rdf:about="#X.509"/>
    </owl:unionOf>
  </owl:Class>
  ...
</owl:Ontology>
```

Figure 1. An illustrative security vocabulary

To verify whether a security constraint, specified according to the defined security vocabulary, is satisfied by a Web service or a Web service composition, we need, in addition to a constraint language, also a language to specify security characteristics of a Web service (referred to as *security capabilities* in what follows). For instance, a security constraint of a Web service provider could require the adoption of a specific authentication mechanism. To verify this constraint, we need to know which authentication mechanisms a Web service supports. Security capabilities describe the security features of a

Web service, according to the specified security vocabulary. We assume that there exists one or more trusted entities in charge of validating and issuing security capabilities.

3.1 Security capabilities

In our framework, Web service security capabilities are expressed through SAML [12] assertions. The SAML architecture relies on the presence of trusted authorities, issuing signed assertions on subjects (e.g., users, services, organizations), that is, a set of statements about the subject.[2] In our approach, we suppose the existence of a Secure Capability Authority (SCA) in charge of evaluating Web service security capabilities, and, based on this evaluation, of issuing signed SAML assertions certifying such capabilities. In particular, we use the attribute statement of SAML assertions to express security capabilities of a Web service, by associating a different attribute with each different Web service security capability. According to the SAML specification, the attribute statement consists of an attribute name and an attribute value. We use the attribute name to denote the security feature, whereas the attribute value gives information on how the security feature is enforced by the corresponding Web service.

```
<saml:AttributeStatement xmlns:sv="#securityVocabulary">
   <saml:Attribute Name ="sv:privacyAccessControl">
  <saml:AttributeValue>
    P3P
 </saml:AttributeValue>
   </saml:Attribute>
</saml:AttributeStatement>
```

Figure 2. An example of security capability

As an example, Figure 2 reports a security capability expressed through attribute assertions. The name of the first attribute is privacy access control, thus denoting the privacy preserving access control mechanism adopted by the Web service. The attribute value is P3P, meaning that the Web service exploits the P3P language to express privacy access control policies. Security capabilities are stored into the WSDL document of the corresponding Web service, by exploiting the extensibility element.

3.2 Security constraints

We classify security constraints into two broad categories, i.e., those specified by the requestor and those that refer to conditions that a Web service can impose to another Web service in order to cooperate with it (referred to as *compatibility constraints).* The first category is further refined into two subcategories: *general* and *specific* constraints. The first refers to those conditions that the Web service requestor states for all the Web services participating to the composition (e.g., adopted privacy or authentication techniques), whereas specific constraints are related to selected Web services within the composition (e.g., the Web service making hotel reservations should use X.509 authentication).

We use a uniform notation to model all types of identified constraints. As for security capabilities, we store compatibility constraints into the WSDL document describing a Web service. More precisely, they are stored into the WSDL extensibility element (see the `Compatibility` element in Figure 3). By contrast, constraints specified by the Web service requestor (i.e., general, and specific constraints) are included into the service request (i.e., in a SOAP message). Security constraints are modeled as Boolean formulas over security capabilities. To make secure matchmaking easier, we store Boolean formulas in a disjunctive normal form, where each clause is modeled by a different sub-element (`Clause` element). The clause element contains the name of the capability to which the condition refers to (`AttributeName` element), the operator of the condition, and the values to be evaluated on that capability (`oper` and `values` element, respectively). Thus, for instance, if a Web service wants to answer only requests of Web services using SAML authentication, or requests that do not use DES encryption, the compatibility constraint stored in its WSDL document is: 'authentication=SAML OR encryption ≠DES', which corresponds to the `Compatibility` element shown in Figure 3.

```
<Compatibility  xmlns:sv="#securityVocabulary">
 <Clause>
    <AttributeName name="sv:authentication"/>
<Oper op="="/>
<Values>
<Value val="sv:SAML"/>
</Values>
 </Clause>
 <Clause>
    <AttributeName name="sv:encryption"/>
<Oper op="≠"/>
<Values>
<Value val="sv:DES"/>
</Values>
 </Clause>
</Compatibility>
```

Figure 3. An example of compatibility constraint

Since constraints must be matched against capabilities issued by a SCA, the broker and the SCAs have to adopt the common reference ontology (shown in Figure 1) to express security capabilities and constraints.

[2] The SAML specification supports three types of statements: *authentication statements*, which assert that a subject has been authenticated by the issuing authority; *authorization statements*, which state that a subject has been given an authorization by the issuing authority; and *attribute statements*, which contain subject information that can be used to grant authorizations.

4 Secure WS-broker

Secure conscious composition of Web services is realized by a Web service, called *Secure WS-Broker* (SWS-Broker for short). The SWS-Broker receives as input a request of a service, whose implementation may require the composition of several Web services. The request contains a description of the requested Web service. Additionally, the SWS-Broker receives a set of general and specific security constraints to be satisfied by the resulting composition. The SWS-Broker first performs the creation of an appropriate workflow (WF) that models the business process generating the required service. This is done with the help of libraries of patterns for well-know business

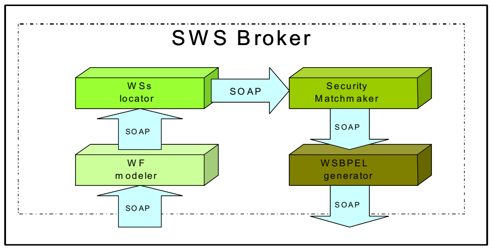


Figure 4. SWS-Broker architecture

processes. This step is deeply affected by the service description given in input. Indeed, the service could be described according to either a syntactic (i.e., WSDL and UDDI) or semantic approach (i.e., DAML-S). Once the appropriate WF has been devised, the SWS-Broker starts generating the composition, which finds out for each WF activity, a suitable Web service. By suitable Web service we mean a Web service having the ability to perform that activity and satisfying the security constraints. The last task performed by the SWS-Broker is the generation of the WSBPEL document representing the secure conscious composition, which is then returned to the requestor. In the case that no secure conscious compositions can be generated (i.e., no suitable Web services are found), the SWS-Broker returns a report containing the security constraints that cannot not satisfied and/or the WF activities for which no Web service have been located.

More precisely, the SWS-Broker consists of four main components (see Figure 4): WF-Modeler, WSs-Locator, Security Matchmaker, and WSBPELgenerator, which we briefly describe in what follows.[3]

WF-Modeler. The SWS-Broker receives as input a service description, referring to the required final Web service. The requestor does not give any direction on how and which Web services should be involved to provide the required service. For this reason, the first step of the SWS-Broker is to model the business process required to produce the requested service. This initial step is done by the WF-Modeler, which returns as a result a workflow. Each activity in the devised WF is complemented by a set of semantic annotations, to describe its functionalities and capabilities.

WSs-Locator: Once the appropriate workflow has been generated, the next step is to identify, for each WF activity, one or more Web services able to carry on the considered activity. This task is performed by the WSs-Locator, which could exploit both UDDI search functionality and semantics annotations to perform the assignment.

Security Matchmaker: The WSs-Locator simply returns for each WF activity a list of Web services able to perform it, without considering any security constraint during this selection. This is done by the Security Matchmaker, which is the core of the SWS-Broker architecture. Indeed, given the WF and Web services returned by the WS-Modeler and the WSs-Locator, respectively, the Security Matchmaker selects, for each WF activity, a Web service satisfying the specified security constraints, among those identified by the WSs-Locator, thus obtaining the secure conscious composition.

WSBPEL Generator: The last step is the translation of the results returned by the Security Matchmaker into a WSBPEL document. The resulting WSBPEL document contains information about the general and specific constraints considered during the security conscious composition. More precisely, these constraints are modeled by means of WS-Agreement [22], and can be exploited for further checks during the execution of the composed web service.

In the following, we describe in details the strategies adopted by the Security Matchmaker, since it represents the core of the proposed solution.

5. Security matchmaker

The goal of the Security Matchmaker is to associate with each WF activity, a Web service satisfying the specified security constraints, among the ones discovered by the WSs-Locator. We recall that security constraints can be of three different types: *general*, *specific* and *compatibility* constraints. General and specific constraints can be verified in an initial phase, by simply pruning from the Web services returned by the WSs-Locator those that do not satisfy the security conditions specified in the constraints. By contrast, compatibility constraints have to be considered during the allocation of a Web service to an activity. This verification process relies on the concept of Web services *compatible with regard to security*. A Web services WS_1 is compatible *with regard to security* with a Web service WS_2 if and only if WS_2 security capabilities satisfy WS_1's compatibilities constraints.

Thus, when selecting a Web service to be associated

[3] The brokered architecture also supports the possibility of delegating some of the tasks to an external and more specialized Web service.

with an activity, the Security Matchmaker has to choose Web services that are *compatible with regard to security* with the Web Services already assigned to activities in the WF preceding the considered one. This matching process is further complicated by the fact that several Web services can be assigned to the same activity, and that the WF may require to execute some activities in parallel. Let us first suppose that the WF consists of a sequence of activities. Moreover, for simplicity, when an activity appears several times in the sequence, we consider each occurrence as a different activity. To perform the security conscious composition, the Security Matchmaker makes use of a data structure, called *composition tree*, representing all possible security conscious Web service compositions. In general, a level *j* of the composition tree is related to the *j-th* activity in the WF, where each node at level *j* represents a Web service able to perform activity *j* and compatible with regard to security with its predecessor nodes. Therefore, each path in the tree denotes a security conscious composition. Given the composition tree associated with a WF consisting of a sequence of activities $\{a_1,\ldots,a_n\}$, the Security Matchmaker selects only the Web service compositions corresponding to complete path of the tree, that is, a path of length *n*.

$$\begin{tikzcd}
 & a_2 \arrow[dr] & & \\
a_1 \arrow[ur] \arrow[dr] & & a_4 \arrow[r] & a_5 \\
 & a_3 \arrow[ur] & &
\end{tikzcd}$$

Figure 5. An example of workflow

In order to better clarify the security matchmaking process, we consider a request of a “travel agency” service able to plan a complete travel consisting of flight and hotel reservations. We suppose that the travel agency takes as input the user request and travel constraints/preferences (e.g., departure and arrival date, preferred flight seat, hotel room types), and finds the best combination of flight plus hotel, based on the user request. Suppose moreover, that the WF associated by the WF Modeler to this request consists of five activities (see Figure 5). Activity a_1 consists of the acquisition of the user travel request (e.g., destination and arrival city, departure and returning dates), and its validation. Activities a_2 and a_3 implement the flight scheduling process. These are two parallel activities that consider two different classes of flights: activity a_2

a_1	WS_2, WS_6, WS_8
a_2	WS_3, WS_4, WS_9
a_3	WS_2, WS_8
a_3	WS_5, WS_7
a_4	WS_{11}, WS_{12}
a_5	WS_{13}, WS_{14}

Table 1. Web Services associated with activities

corresponds to a flight scheduling service that considers only direct connections, whereas activity a_3 considers only low cost companies. Activity a_4 is the hotel scheduling. Finally, during the last activity, i.e., a_5, all the returned combinations of flight plus hotel are evaluated in order to find the best one, according to the user request.

In order to explain the security matchmaking process, let us start to consider a single sequence of activities in the WF represented in Figure 5, that is, the sequence consisting of activities a_1, a_2, a_4, and a_5. Suppose, moreover, that the corresponding Web services, identified by the WSs-Locator are those in Table 1.

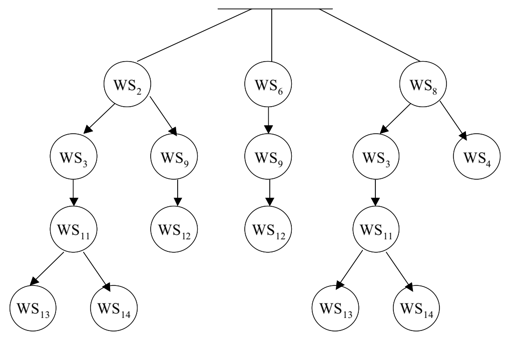


Figure 6. An example of composition tree

The composition tree built by the Security Matchmaker for this sequence is represented in Figure 6. In particular, the first level of the tree contains all the Web services performing activity a_1 (i.e., WS_2, WS_6, and WS8). To build the second level of the tree, the Security Matchmaker verifies the compatibility with regard to security among Web services associated with activity a_1 and Web services associated with activity a_2 (i.e., WS_3, WS_4, and WS_9). The composition tree represented in Figure 6 has been built by supposing that WS_2 is compatible only with WS_3 and WS_9, whereas WS_6 is compatible only with WS_9, and WS_8 with WS_3 and WS_4. Moreover, we have supposed that WS_{11} is compatible only with WS_{13} and WS_{14}, whereas WS_9 with WS_{12}. Once the composition tree has been completed, the Security-Matchmaker is able to determine the corresponding secure conscious Web service compositions, which, according to Figure 6, are: $\{WS_2, WS_3, WS_{11}, WS_{13}\}$, $\{WS_2, WS_3, WS_{11}, WS_{14}\}$, $\{WS_8, WS_3, WS_{11}, WS_{13}\}$, and $\{WS_8, WS_3, WS_{11}, WS_{14}\}$. In the syntax of transaction logic [23], the output of the Security Matchmaker can be represented as follows:

workflow $\leftarrow$ **START** $\otimes$ **(WS_2 | WS_8)** $\otimes$ **WS_3** $\otimes$ **WS_{11}** $\otimes$ **(WS_{13} | WS_{14})** $\otimes$ **END**

where symbol “⊗” means serial conjunction between two activities and “|” means concurrent conjunction for two or more activities.

In general, a WF (see Figure 5) consists of more complex structures than activities sequences (e.g., parallel activities, while and branch conditions). In order to apply the above-introduced approach to a more complex workflow, the Security Matchmaker first derives the set of activity sequences associated with it. Note that the activity sequences associated with a workflow always have some common activities (at least the activity starting the workflow). As an example, considering again the workflow in Figure 5, a_1, a_4, and a_5 are examples of common activities. The Security Matchmaker derives from the WF two sequences: $\{a_1, a_2, a_4, a_5\}$ and $\{a_1, a_3, a_4, a_5\}$. In the syntax of transaction logic, this workflow in terms of activities can be represented as follows:

$$\mathbf{workflow} \leftarrow \mathbf{a_1} \otimes (\mathbf{a_2} \mid \mathbf{a_3}) \otimes \mathbf{a_4} \otimes \mathbf{a_5}$$

Then, it builds the composition tree for each of the derived sequences, according to the strategy described above, and selects a secure conscious composition for each sequence, according to several criteria (e.g., resources optimization, best quality of service average). The selected secure conscious composition must have the common activities assigned to the same Web service. For illustration, the simplified WSBPEL for Figure 5 is shown in Figure 8.

The WSBPEL generator inserts information about security constraints into the resulting WSBPEL document. This information can be exploited for further runtime checks during web service execution. More precisely, security constraints are modeled by WS-Agreement (e.g., namespace wsag) [22]. WS-Agreement is used to define the capabilities of service providers and create agreements based on creational offers for monitoring agreement compliance at runtime. In particular, we use the WS-Agreement free-form constraint assertions "Creation Constraints" to express the constraints. Figure 7 shows an illustrative constraint: "It is required to adopt P3P to tackle the privacy issue."

```
<wsag:template xmlns:sv="#securityVocabulary">
  <wsag:CreationConstraints>
    <wsag:Item>sv:privacyAccessControl</wsag:Item>
    <wsag:Constraint>P3P</wsag:Constraint>
</wsag:CreationConstraints>
</wsag:template>
```

Figure 7. An example of security constraint

The WSBPEL generator inserts into each Web service's invocation a different WS-agreement node for each different general constraint, for each specific constraint applied on that Web service, and for each compatibility constraint that the Web service has to satisfy in order to join the composition. Let us assume for instance that the secure conscious composition has been required with the following constraints: all Web services must exploit SAML as authentication framework, whereas only Web services that manage user's credit card info must use TDES encryption. Moreover, let us assume that according to the workflow in Figure 5, the only activity that handles the credit card information is a_5. Each Web service's invocation of the resulting WSBPEL document (see Figure 8 for a simplified version) is complemented with a WS-agreement node modeling the general authentication requirement (i.e., SAML). By contrast, the specific encryption requirement (i.e., TDES) has been inserted only into the invocation of Web services associated with activity a_5, that is, WS13 and WS14. Regarding compatibility constraints, for simplicity let us consider only the constraints between WS2 and WS3, and assume that in order to allow a composition, WS2 requires to compose only with P3P-enabled web services. In such a case, the WSBPEL generator inserts an additional WS-agreement node modeling WS2's privacy requirements into the WS3's invocation of the resulting WSBEPL document.

```
<process name="workflow"
      targetNamespace="http://travelagencies.com/workflow"
      xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
      xmlns:wsag="http://schemas.ggf.org/graap/2005/09/ws-agreement"
      xmlns:sv="#securityVocabulary"
      suppressJoinFailure="yes">
  ...
  <flow>
    ...
    <receive partnerLink="START"
           portType="ta:InitiateWS_IF"
           outputVariable="request">
      <source linkName="START-to-WS2"/>
      <source linkName="START-to-WS8"/>
    </receive>
    <invoke  partnerLink="WS2"
           portType="as:WS2_IF">
           inputVariable="request"
           outputVariable="responseWS2">
      <target linkName="START-to-WS2"/>
      <source linkName="WS2-to- WS3"/>
      <wsag:CreationConstraints>
        <wsag:Item>sv:authentication</wsag:Item>
        <wsag:Constraint>SAML</wsag:Constraint>
      </wsag:CreationConstraints>
    </invoke>
    <invoke  partnerLink="WS8"
           portType="as:WS8_IF"
           inputVariable="request"
           outputVariable="responseWS8">
      <target linkName="START-to-WS8"/>
      <source linkName="WS8-to-WS3"/>
      …

   <wsag:template xmlns:sv="#securityVocabulary">
     <wsag:CreationConstraints>
       <wsag:Item>sv:privacy</wsag:Item>
      <wsag:Constraint>P3P</wsag:Constraint>
</wsag:CreationConstraints>
   </wsag:template>

     <wsag:CreationConstraints>
        <wsag:Item>sv:authentication</wsag:Item>
        <wsag:Constraint>SAML</wsag:Constraint>
     </wsag:CreationConstraints>
    </invoke>
    <invoke  partnerLink="WS3"
            portType="as:WS3_IF"
            inputVariable="responseWS2| responseWS8"
            outputVariable="responseWS3">
      <target linkName="WS2-to-WS3"/>
      <target linkName="WS8-to-WS3"/>
      <source linkName="WS3-to-WS11"/>
      …
   <wsag:CreationConstraints>
        <wsag:Item>sv:authentication</wsag:Item>
        <wsag:Constraint>SAML</wsag:Constraint>
      </wsag:CreationConstraints>
    </invoke>
    <invoke  partnerLink="WS11"
            portType="as:WS11_IF"
            inputVariable=" responseWS3"
            outputVariable="responseWS11">
      <target linkName="WS3-to-WS11"/>
      <source linkName="WS11-to-WS13"/>
      <source linkName="WS11-to-WS14"/>
      …
  <wsag:CreationConstraints>
        <wsag:Item>sv:authentication</wsag:Item>
        <wsag:Constraint>SAML</wsag:Constraint>
      </wsag:CreationConstraints>

    </invoke>
    <invoke  partnerLink="WS13"
            portType="as:WS13_IF"
            inputVariable="responseWS11"
            outputVariable="responseWS13">
      <target linkName="WS11-to-WS13"/>
      <source linkName="WS13-to-END"/>
   …
   <wsag:CreationConstraints>
        <wsag:Item>sv:encryption</wsag:Item>
        <wsag:Constraint>TDES</wsag:Constraint>
      </wsag:CreationConstraints>
     <wsag:CreationConstraints>
        <wsag:Item>sv:authentication</wsag:Item>
        <wsag:Constraint>SAML</wsag:Constraint>
      </wsag:CreationConstraints>
    </invoke>
    <invoke  partnerLink="WS14"
            portType="as:WS14_IF"
            inputVariable="responseWS11"
            outputVariable="responseWS14">
      <target linkName="WS11-to-WS14"/>
      <source linkName="WS14-to-END"/>
      …
<wsag:CreationConstraints>
        <wsag:Item>sv:encryption</wsag:Item>
        <wsag:Constraint>TDES</wsag:Constraint>
      </wsag:CreationConstraints>
  <wsag:CreationConstraints>
        <wsag:Item>sv:authentication</wsag:Item>
        <wsag:Constraint>SAML</wsag:Constraint>
      </wsag:CreationConstraints>
    </invoke>
    <reply partnerLink="END"
          portType=" ta:ReplyWS_IF "
          inputVariable="responseWS13| responseWS14" />
      <target linkName="WS13-to-END"/>
      <target linkName="WS14-to-END"/>
    </reply>
    ...
  </flow>
</process>.
```

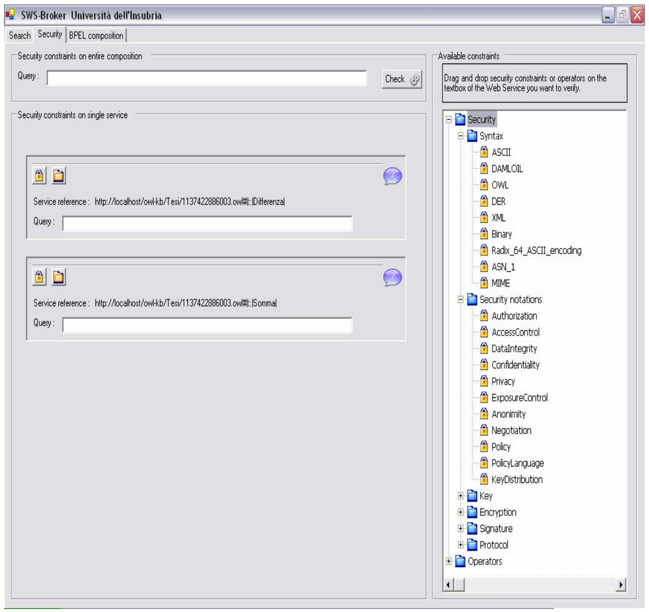
Figure 8. A Simplified WSBPEL Document

6. Prototype implementation

We have implemented a prototype of the SWS-Broker proposed in this paper. The SWS-Broker has been implemented as a Web service. The prototype has been developed using the NetBeans 5.0 environment. The GUI has been implemented in C#. Constraint matching is done by using the JTP engine [24]. At the current stage, the prototype is able to manage only sequential WFs. However, we plan to extend it to deal with more general WFs. The GUI allows Web service requestors to inquire the SWS-Broker for the needed services. By using the GUI, the requestor can also specify constraints that the resulting Web service (or some of its components) must satisfy. The interface for constraint specification is presented in Figure 9. Once the constraints have been specified and the WF Modeller has generated the WF, the SWS-Broker starts building the secure conscious composition. For each considered Web service, it extracts compatibility constraints from its WSDL document and matches them against the security capabilities of the Web services already belonging to the composition (using the JTP engine as core component). Similar matches are performed for general and specific constraints. Due to efficiency reasons the prototype does not built the whole composition tree, rather it uses a heuristics according to which the tree is built using a depth first strategy. Therefore, as soon as the first complete path has been built, the corresponding WSBPEL document is generated and returned to the requestor.

7. Conclusions

In this paper, we have tackled the problem of Web service composition, focusing on security issues. We proposed an approach to compose Web services according to specified security requirements of both Web service requestors and providers. This work is just a first step of a wider project we are currently working on. First, we plan to extend our proposal to other classes of constraints (such as for instance quality of services constraints). We plan also to extend the proposed approach by considering privacy of security constraints and capability.



Drag and drop security constraints or operators on the textbox of the Web Service you want to verify.

Service reference : http://localhost/owl-kb/Tesi/1137422886003.owl#|::|Differenza|
Query :

Service reference : http://localhost/owl-kb/Tesi/1137422886003.owl#|::|Somma|
Query :

Figure 9. Constraint specification

Moreover, we plan to devise efficient techniques for the generation of composition trees, in order to minimize the number of paths to be computed. Finally, we plan to integrate the current proposal with the work reported in [4], which provides a solution to privacy issues related to Web services discovery agencies.

References

[1] Ankolekar, A., Burstein, M., Hobbs, J.R., Lassila, O., Martin, D.L., McIlraith, S.A., Narayanan, S., Paolucci, M., Payne, T., Sycara K., and Zeng, H. 2001. DAML-S: Semantic Markup For Web Services. *International Semantic Web Working Symposium (SWWS)*, Standford University, CA, USA.
[2] Arkin. A. 2002. Business Process Modeling Language (BPML), Version 1.0. BPMI.org..
[3] Bultan, T., Fu, X. Hull, R., and Su, J. 2003. Conversation Specification: A New Approach to Design and Analysis of E-Service Composition. *Twelth Intl. World Wide Web Conference (WWW2003).*
[4] Carminati, B., Ferrari, E., and Hung P.C.K. Exploring Privacy Issues in Web Services Discovery Agencies. *IEEE Security & Privacy Magazine*, 3(5): 14-21, 2005..
[5] IBM Corporation. 2002. Business Process Execution Language for Web Services (BPEL4WS), Version 1.0.
[6] IBM, Microsoft and VeriSign. 2002. Specification: Web Services Security (WS-Security), Version 1.0, 05 April 2002.
[7] IBM, BEA, Microsoft, SAP, Sonic Software, VeriSign. 2004. Web Services Policy Framework (WS-Policy), September 2004.
[8] Kagal, L., Finin, T., Joshi, A. 2003. A Policy Based Approach to Security on the Semantic Web, *The Semantic Web – ISWC..*
[9] Kagal, L., Paolucci, M., Srinivasan, N., Denker, G.,. Finin, T., Sycara, K. 2004. Authorization and Privacy for Semantic Web Services, *AAAI Spring Symposium, Workshop on Semantic Web Services.*
[10] McIlraith, S., Son, T.C. 2002. Adapting Golog for Composition of Semantic Web Services *8th International Conference on Knowledge Representation and Reasoning (KR2002).*
[11] Medjahed, B., Bouguettaya, A., and Elmagarmid, A.K. 2003. Composing Web Services on the Semantic Web. *The VLDB Journal*, 12(4).
[12] OASIS. SAML 1.0 Specification Set:, 2002.
[13] OASIS. Web Services Business Process Execution Language (WSBPEL).
[14] Universal Description, Discovery and Integration (UDDI). 2002. UDDI v. 3.0, UDDI Spec Technical Committee Specification.
[15] World Wide Web Consortium (W3C). 2002. Web Services Description Language (WSDL), Version 1.2, W3C Working Draft.
[16] World Wide Web Consortium (W3C). 2003. SOAP Version 1.2 Part 1: Messaging Framework, W3C Proposed Recommendation,.
[17] OASIS. 2005. OASIS extensible access control markup language (XACML). Version 2.0, OASIS Standard. Online: http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.
[18] World Wide Web Consortium (W3C). The platform for privacy preferences 1.0 (P3P1.0) specification. W3C Recommendation. Retrieved on April 16, 2002. Online: http://www.w3.org/TR/P3P/.
[19] IBM. Enterprise Privacy Authorization Language (EPAL). IBM Research Report. Retrieved June 12, 2003. Online: www.zurich.ibm.com/security/enterprise-privacy/epal
[20] Hung, P. C. K., Ferrari, E., and Carminati, B. Towards Standardized Web Services Privacy Technologies. In Proceedings of the 2004 IEEE International Conference on Web Services (ICWS'04), San Diego, California, USA, July 6-9, 2004.
[21] WebOnt. 2003. OWL Web Ontology Language. Web-Ontology (WebOnt) Working Group, World Wide Web Consortium (W3C). Online: http://www.w3.org/2001/sw/WebOnt.
[22] Grid Resource Allocation Agreement Protocol (GRAAP) WG. 2005. Web Services Agreement Specification (WS-Agreement)., Version 2005/09, GWD-R (Proposed Recommendation).. Online: http://www.ggf.org/Public_Comment_Docs/Documents/Oct-2005/WS-AgreementSpecificationDraft050920.pdf
[23] Bonner, A.J. 1999. Workflows, Transactions, and Datalog. In Proceedings of the 18th ACM Symposium on principles of Database Systems (PODS), 294-305.
[24] Richard, F., Jenkins, J., and Frank, G. 2003. JTP: A System Architecture and Component Library for Hybrid Reasoning. *Proceedings of the Seventh World Multiconference on Systemics, Cybernetics, and Informatics.* Orlando, Florida, USA. July 27 - 30, 2003.