Security Conscious Web Service Composition with Semantic Web Support[1]

Barbara Carminati[*], Elena Ferrari[*], Ryan Bishop[°], Patrick C. K. Hung[°]
[*]*University of Insubria, Department of Computer Science and Communication Italy*
[°]*University of Ontario Institute of Technology (UOIT), Canada*
{barbara.carminati, elena.ferrari}@uninsubria.it
ryan.bishop@mycampus.uoit.ca; patrick.hung@uoit.ca

Abstract

A Web service is a software system designed to support interoperable application-to-application interactions over the Internet. Recently, there has been a growing interest in Web service composition, and some languages (e.g., WSBPEL and BPML) for modeling the composition have been proposed. In this paper, we focus on security constraints of Web service composition with semantic Web support, which have not been deeply investigated so far. Based on our prior work, we present a method for modeling security constraints and a brokered architecture, which exploits the REI reasoner, to build composite Web services according to the specified security constraints.

Keywords: Semantic Web, Web service composition, security constraints, REI.

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) [23], Web Services Description Language (WSDL) [25], and Simple Object Access Protocol (SOAP) [26]. 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 [9], WSBPEL [21] and BPML [2]) have been proposed to coordinate Web services into a workflow. A workflow is a computer supported business process.

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 [10] to ensure messages have not been modified or forged while in transit or while residing at destinations.

In this paper, we consider a security aspect of Web service composition that has not been so far deeply investigated, despite its importance, that is, the one related to security requirements to be considered in composing Web services. 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 compositions driven by security requirements as *security conscious compositions*. 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. To model and enforce such constraints and security properties we exploit Semantic Web technologies. The Semantic Web comprises the standardization and use of descriptive technologies (e.g., RDF, OWL) to relate data on the Web across systems in various languages, as an enhancement of the World Wide Web. For example, semantic Web languages could be used to define ontology. However, there is still no integration framework to link among Web service composition, semantic Web and security requirements.

We first propose a way to model security constraints to be considered during Web service composition, which is compliant with existing Web

[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" with the NSFC grants no. 60473091 and 60673175 in China.

1-4244-0832-6/07/$20.00 ©2007 IEEE.

service standards. In particular, we use the REI language to define security constraints and the REI policy engine [14] as a reasoning system to reason over security constraints for matchmaking. Then, we present a semantic brokered architecture to compose Web services according to the specified security constraints, which exploits the REI policy engine.

The work reported in this paper builds on a system for secure conscious composition of Web services proposed by us in [6]. Differently from [6], in the current paper we exploit the use of semantic techniques to model security constraints and we show how the brokered architecture proposed in [6] on support of secure conscious composition can be modified to cope with them.

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 with Semantic Web support. Section 4 describes the architecture on support of security conscious Web service composition; whereas Section 5 presents an example of secure conscious composition. Finally, Section 6 concludes the paper and outlines directions for future work.

2. Related work

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 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 [21]. 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 UDDI registries, 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 [16], 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). A basic difference between the approach reported in [16] 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 [16] they use a semantic annotation-based approach. Moreover, our approach to model the security capabilities of a Web service is aligned with the conceptual model of Security Assertions Markup Language (SAML). A further relevant difference is that in [16] the authors only consider the enforcement of security constraints of a single Web service requester. By contrast, in the proposed approach we consider security requirements of both Web service requestors and Web services taking part in the composition.

Sycara et al. extend OWL-S to model constraints and capabilities of Web services [22]. In their approach, the REI language is exploited in the context of Web service. However the REI language does not address the issues of matching security constraints and capabilities in the Web service composition. For instance, a Web service may allow the exchange of information without encryption, while another Web service in the workflow may require encryption. Sycara et al. only focus on the encoding of security information in the inputs and outputs of the OWL-S process description [22]. Referring to our approach, we exploit a security matchmaker to verify Web services compatibility with regard to security requirements of both Web service requestors and providers.

Bartoletti et al. propose a formal framework for statically determining a secure plan, i.e., a secure composition [3] of Web services. In particular, in their approach each Web service has associated an abstract high level description of its behaviour. Then, model-checking techniques are exploited to verify the validity wrt security of the overall behaviour produced by a plan. Authors consider three kinds of plans: simple plans, selecting one service for each request, multi-choice plans, choosing among a set of services for each request, and dependent plans, which exploit the knowledge of the past choices to select the best

combination. With respect to our approach, we have proposed a brokered architecture with the aim to be compliant as much as possible to Web standards, making at the same time use of semantic technologies. Moreover, our secure matchmaker has no limitations on the possible plans, in that all possible workflows can be validated.

Other related work are those exploiting AI planning techniques for Web service composition. Among them, we recall the work by McIlraith et al. [18] that extends the logic programming language Golog for automatic composition of Web services, the one by Medjahed [19], 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. [4]. However, such frameworks do not address security issues, 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 SAML, which is used to define authentication and authorization decisions in Web services [20]. Web services providers submit SAML tokens to security servers for making security decisions. Another XML-based language is the one proposed in WS-Security, which describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality and single message authentication [10]. Based on WS-Security, WS-Policy provides a grammar for expressing Web services security policies [11]. The WS-Policy includes a set of security policy assertions to support the WS-Security specification defined in WS-Security Policy [11].

Our proposal exploits the REI policy language. Kagal, in developing a policy language for the Me-Centric project, [12] identifies that most research in policy languages has been narrowed to specific application domains, i.e., security, system management. Thus, there are no general specifications for policy verification. Also, the integration of the Semantic Web must be supported by the policy language. To overcome these drawbacks, Kagal in [12] developed the REI policy language as a flexible language for policy specification and reasoning across multiple domains to be applied in the Me-Centric project. REI allows security, management, and conversational policies to be described by four basic constructs. The four basic constructs of REI are: rights, obligations, dispensations, and prohibitions [15]. Rights are the permissions an entity has to perform the associated actions. A permission allows an agent to perform associated actions if and only if certain constraints are satisfied [13]. Obligations are actions triggered by conditions that an entity must perform. Dispensations are waivers of obligations so an entity no longer has to perform an action. Prohibitions are negative authorizations stating that an entity can not perform an action. REI policies allow the specification of privacy, authentication, and confidentiality requirements for Web service providers and requesters [17]. After extensive research, we found that the REI policy language is the best policy language that meets the requirements of secure conscious Web service composition discussed in this paper. The authors in [7] propose an augmented distributed trust model built upon a public key infrastructure that provides authentication, non-repudiation, anti-playback, and access control. The work extends Project Centaurus, responsible for maintaining and executing services at users' requests on an independent infrastructure to reduce the load on portable devices, by deployment of decentralized services in the system. The primary goal of the system is to support users' remote access to services across domains in a simple yet secure way. The system uses Prolog to model permissions, obligations, entitlements, and prohibitions. Moreover, rights may be delegated to sustain remote access to services if a particular user's role does not provide permission. We consider the importance of deferring access rights so long as security remains paramount.

3. Security capabilities and constraints

Grounded in Resource Description Framework (RDF) and OWL, REI allows the definition of declarative policies over a security vocabulary and other domain specific ontologies. In our approach, the REI policy language is used to define security constraints of both Web services and Web service requestors. Also, we use REI as a reasoning system to reason over security constraints for matchmaking. In addition, our approach integrates the REI-Reasoner into the Web services architecture [27].

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) [24]. 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 standards.

```
<owl:Ontology rdf:about="#securityVocabulary">
 <owl:versionInfo>v1.00 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="#REI"/>
   <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 in OWL

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, as security constraint a Web service provider could require the adoption of a specific authentication mechanism such as X.509. 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 [20] 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: AccessControl">
          <saml:AttributeValue>
          XACML
          </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 attribute is privacy access control, thus denoting the access control mechanism adopted by the Web service. The attribute value is XACML, meaning that the Web service exploits the XACML language to express 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

In theory, there are three types of constraints: (1) *Static constraints*, that can be evaluated without executing the Web service composition; (2) *Dynamic 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.

that can be evaluated only during the execution of a Web service composition, because they express restrictions based on the execution history of the Web service composition; and (3) *Hybrid constraint*s, that is, constraints whose satisfiability can be partially verified without executing the Web service composition. This paper will cover static and dynamic constraints.

We further 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).

```
<constraint:CompatibilityConstraint
        xmlns:sv="#securityVocabulary"
        rdf:ID="compatibilityConstraint1" type="static">
   <constraint:subject rdf:resource="sv:authenticaion"/>
    <constraint:predicate rdf:resource="&operator:equal"/>
    <constraint:object rdf:resource="sv:SAML"/>
</constraint: CompatibilityConstraint>
<constraint:CompatibilityConstraint
        xmlns:sv="#securityVocabulary"
        rdf:ID="compatibilityConstraint2" type="static">
  <constraint:subject rdf:resource="sv:encryption"/>
  <constraint:predicate rdf:resource=" &operator:notEqual"/>
  <constraint:object rdf:resource="sv:DES"/>
</constraint: CompatibilityConstraint>
```

Figure 3. An example of compatibility constraint

We use a uniform notation to model static and dynamic types of identified constraints (see the `Type` attribute in Figure 3). 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 `CompatibilityConstraint` 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 element (`CompatibilityConstraint` element). The compatibility constraint element contains the name of the capability to which the condition refers to (`subject` element), the operator of the condition, and the values to be evaluated on that capability (`predicate` and `object` 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 `CompatibilityConstraint` element shown in Figure 3.

Referring to Figure 3, both constraints are classified as static constraints. 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.

Figure 4 shows an example of general/static constraint defined in REI. This constraint states that the Web service requestor requires that all Web services participating to the composition exploit the X.509 framework for authentication. This static constraint is handled by the matchmaker before the execution of Web service composition.

```
<constraint:SimpleConstraint
        xmlns:sv="#securityVocabulary"
        rdf:ID="GeneralConstraint1" type="static">
   <constraint:subject rdf:resource="sv:authenticaion"/>
   <constraint:predicate rdf:resource="&operator:adopt"/>
   <constraint:object rdf:resource="sv:X.509"/>
</constraint:SimpleConstraint>
```

Figure 4. An example of general constraint

By contrast, Figure 5 shows an example of specific/dynamic constraint defined in REI. This constraint states that an authentication request for connecting to Web service 1 (WS1) can not fail more than three times. In this case, this dynamic constraint can not be handled by the matchmaker alone before the execution of Web service composition. This constraint has to be handled and monitored by the workflow executor during the execution of Web service composition.

```
<constraint:SimpleConstraint
              xmlns:esv="#enhancedSecurityVocabulary"
              rdf:ID="SpecificConstraint1" type="dynamic">
 <constraint:subject
          rdf:resource="esv:AuthenticationRetry"/>
 <constraint:predicate
          rdf:resource="&logicaloperator:lessThan"/>
  <constraint:object rdf:resource="&times;3"/>
</constraint:SimpleConstraint>
```

Figure 5. An example of specific constraint

Constraints (general, specific and compatibility) can be combined in pairs using the Boolean operators *And*, *Or*, and *Not,* forming more complex constraints, called *composite constraints*. For instance, the constraints that require that all Web services participating to the composition exploit X.509 framework for authentication, *and* an authentication request for connecting to Web service 1 can not fail more than 3 times, are paired in Figure 6. In this case, the composite constraint combines a specific and a general constraint. This constraint can be partially verified before executing the composition, since the general constraint component is static, whereas the component referring to the specific constraint can be only evaluated at run-time.

```
<constraint:And rdf:ID="GeneralAndSpecificConstraint">
      <constraint:first rdf:resource="#GeneralConstraint1"/>
      <constraint:second
          rdf:resource="#SpecificConstraint1"/>
</constraint:And>
```

Figure 6. An example of composite constraint

4. Secure WS broker

In what follows, we describe how the brokered architecture proposed in [6] for secure conscious composition of Web services can be extended to support constraints modeled using REI.

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 (both static and dynamic) 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-known business 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 be satisfied and/or the WF activities for which no Web service have been located.

Security constraints verification is done by adapting the REI's policy reasoning engine. The REI-Reasoner is integrated into the security matchmaker to evaluate the compatibility of constraints and capabilities of different Web services. The REI-Reasoner and Security Matchmaker work in unison to select the services which best meet the original request. More precisely, the SWS-Broker consists of five main components and the interactions are formatted in SOAP (see Figure 7): WF-Modeler, WSs-Locator, Security Matchmaker, REI-Reasoner, WSBPEL Generator and WSBPEL Executor, which we briefly describe in what follows.[3]

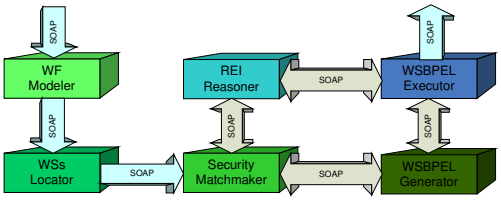

Figure 7. SWS-Broker Architecture

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

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 in WSBPEL. We propose to use workflow technologies such as Workflow Management System (WFMS) in WF-Modeler. WFMS is the software to support the specification, decomposition, execution, coordination, and monitoring of workflows [28]. 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. Both matchmaking and delegation processes are expected to use the UDDI to find the most appropriate Web service to satisfy WF activities' or even sub-activities' requirements.

Security Matchmaker: The WSs-Locator simply returns for each WF activity a list of Web services able to perform it, without considering static and dynamic security constraints and compatibility constraints 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. In this scenario, the role of WSs-Locator is to assign an appropriate Web service for each activity. This assignment process is called matchmaking. Furthermore, value-added Web services are required to be enacted by long duration multi-step activities. Thus Web services may also delegate some sub-activities that are decomposed from the assigned activities to other Web services. This assignment process is called delegation.

REI-Reasoner: The Security Matchmaker sends the security constraints with the selected Web service in OWL-S description to the REI-Reasoner. In this stage, static general and specific constraints are resolved by the REI policy engine in cooperation with the Security Matchmaker. The Security Matchmaker will only select the Web service for an activity that can satisfy static constraints of that specific activity. The policy engine is structured around the meta policies of the requestor to resolve conflicts that force constraints to be accepted or rejected by the system. Furthermore, the REI policy engine reasons about the selected Web service to monitor and ensure that static compatibility constraints are not violated. The policy engine can only reason about policies that it has knowledge of. However, REI supports both domain dependent and independent information. Thus, REI provides specifications for representing domain independent information, allowing the engine to reason over information that is not included in the knowledge base. For dynamic constraints, the REI-Reasoner works with the WSBPEL Executor and the Security Matchmaker. Finally, the REI-Reasoner passes the results to the Security Matchmaker that forwards them to the WSBPEL Generator.

WSBPEL Generator: The last step is the translation of the results returned by the Security Matchmaker into a WSBPEL document that includes the specification and decomposition of workflows [28]. Again this step involves WFMS. 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 [8], and can be exploited for further checks during the execution of the composed Web service. WS-Agreement is a service-level agreement (SLA), which represents a formal contract between a Web services requestor and provider guaranteeing quantifiable issues at defined levels only through mutual concessions. More details on the WSBPEL Generator can be found in [6].

WSBPEL Executor: The last step is the execution, coordination and monitoring of workflows [28]. Again this step involves WFMS as a middleware to support Web services execution and management in according to the WSBPEL document. The WSBPEL executor component is also called the run-time engine which consists of an execution end-user interface. The run-

time engine is an execution environment which assists or performs the coordination of WSBPEL.

5. An illustrative example

In this section, we present an example to clarify how a secure conscious composition is generated by SWS-Broker. A detailed description of the Security Matchmaker can be found in [6]. In doing this, let us assume that a user requires to the SWS-Broker a "travel plan" service, by which to plan a complete travel consisting of flight, hotel, and car reservations. Moreover, let us assume that the requestor specifies a set of constraints. More precisely, he/she requires that all Web services participating to the composition exploit the X.509 framework for authentication (i.e., a general constraint). Additionally, the Web service carrying out the hotel reservation (A2) must adopt XACML for access control policy specification (i.e., a specific constraint). Both of these constraints are static. In addition, the Web service requestor specifies a dynamic security constraint stating that the number of authentication for a request to the car reservation activity can not fail more than three times. In order to create the secure conscious composition, the SWS-Broker first inquires the WF modeler for modeling such a service.

Figure 8. The workflow for a travel planning

By assuming that the workflow returned by the modeler is the one reported in Figure 8, the SWS-Broker has to find out for each of the activities depicted in Figure 8 one or more Web services able to carry out them. This task is performed by WS-Locator, which returns for each WF activity the WSDL documents of those Web services able to perform the corresponding activity. More precisely, we assume that WS-Locator found out Web services WS1, WS2, and WS3 for activity A1, Web services WS4 and WS5 for activity A2, and Web services WS6 and WS7 for activity A3 (see Table 1). Once the Web services have been located, the SWS-Broker can evaluate the security constraints, that is, the requestor constraints and compatibility constraints of the discovered Web services.

Activity	Web Services	Web Services Capabilities
A1	WS1	Authentication=SAML
	WS2	Authentication=X.509
	WS3	Authentication=SAML
A2	WS4	Authentication=X.509 AccessControl=XACML
	WS5	Authentication=X.509 AccessControl=XACML Signature=XML-SIG
A3	WS6	Authentication=X.509
	WS7	Authentication=SAML

Table 1. Security capabilities of Web services returned by WS-Locator

As introduced in Section 4, this task is performed by the Security Matchmaker, which separately evaluates both kinds of security constraints with the help of the REI engine. In particular, it starts to evaluate the requestor constraints. In doing that, from each WSDL document returned by the WS-Locator it extracts the security capabilities of the corresponding Web service. Table 1 shows the results of this step, where with each Web service it is associated its security capabilities.

During the evaluation of requestor constraints, the Security Matchmaker prunes from the Web services returned by WF-Locator, those that do not satisfy the constraints specified by the Web service requestor. This implies pruning from all Web services those that do not adopt X.509. Moreover, only from the Web services associated with activity A2 (i.e., hotel reservation), it has to remove those Web services that do not adopt XACML. The final result of the requestor constraints evaluation is presented in Table 2, which reports also the corresponding compatibility constraints.

Activity	Web Service	Web Service Capabilities	Web Service Compatibility Constraints
A1	WS2	Authentication=X.509	Signature= XML-SIG
A2	WS4	Authentication=X.509 AccessControl= XACML	-
	WS5	Authentication=X.509 AccessControl= XACML Signature=XML-SIG	-
A3	WS6	Authentication=X.509	-

Table 2. Security capabilities and compatibility constraints of Web services after the evaluation of requestor constraints

Activity	Web Service	Web Service Capabilities	Web Service dynamic constraints
A1	WS2	Authentication=X.509	
A2	WS5	Authentication=X.509 AccessControl= XACML Signature=XML-SIG	-
A3	WS6	Authentication=X.509	Fail(Authentication) < 3

Table 3. The secure conscious composition

In the second step, the Security Matchmaker evaluates compatibility constraints. To do that, it extracts from the WSDL documents the compatibility constraints specified by Web services. As reported in Table 2, there is only one Web service having a compatibility constraint, i.e., WS2, who requires to the consequent Web services to adopt XML-SIG standard as signature mechanism. In order to satisfy this constraint, the SWS-Broker prunes from the Web services associated with activity A2 those that do not adopt XML-SIG. Table 3 reports the result of the evaluation of compatibility constraints and also the dynamic constraint on A3 (which will be handled by the REI-Reasoner and WSBPEL Executor).

Finally, the secure conscious composition resulting after constraint evaluation is translated into a WSBPEL document (see [6] for more details).

6. Conclusions

In this paper, we have tackled the problem of Web service composition, focusing on security issues. We have proposed an approach to compose Web services according to specified security requirements of both Web service requestors and providers which uses REI to model security constraints and reasoning on them.

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 hybrid or quality of services constraints). We plan also to extend the proposed approach by considering privacy of security constraints and capabilities. Moreover, we plan to devise efficient techniques for generating Web service compositions. Up to now [6] we use a naïve strategy that basically considers all the possible combination of Web services among those selected by the WS-Locator, until it finds one that satisfies the specified security constraints.

Finally, we plan to integrate the current proposal with the work reported in [5], which provides a solution to privacy issues related to Web services discovery agencies.

7. 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. DAML-S: Semantic Markup For Web Services. In *Proceedings of the International Semantic Web Working Symposium (SWWS)*, Standford University, USA, 2001.

[2] Arkin. A. Business Process Modeling Language (BPML), Version 1.0. BPMI.org. , 2002.

[3] Bartoletti, M., Degano, P. and Ferrari, G.L. Plans for service composition. In *Proceedings of the Workshop on Issues in the Theory of Security (WITS)*, Vienna, Austria, 2006.

[4] Bultan, T., Fu, X. Hull, R., and Su, J. Conversation Specification: A New Approach to Design and Analysis of E-Service Composition. In *Proceedings of the Twelfth Intl. World Wide Web Conference (WWW2003)*, Budapest, Hungary, 2003.

[5] 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.

[6] B. Carminati, E. Ferrari, P.C.K. Hung. Secure Conscious Web Service Composition. In *Proceedings of the IEEE International Conference on Web Services*. Chicago, USA, 2006.

[7] A. Cedilnik, L. Kagal, F. Perich, J. Undercoffer, A. Joshi. A Secure Infrastructure for Service Discovery and Access in Pervasive Computing. *ACM Monet: Special Issue on Security in Mobile Computing Environments,* 8(2), 2003.

[8] Grid Resource Allocation Agreement Protocol (GRAAP) WG. Web Services Agreement Specification (WS-Agreement)., Version 2005/09, GWD-R (Proposed Recommendation).Online:http://www.ggf.org/Public_Comment_Docs/Documents/Oct-2005/WSAgreementSpecification Draft050 920.pdf, 2005.

[9] IBM Corporation. Business Process Execution Language for Web Services (BPEL4WS), Version 1.0., 2002.

[10] IBM, Microsoft and VeriSign. Specification: Web Services Security (WS-Security), Version 1.0, 2002.

[11] IBM, BEA, Microsoft, SAP, Sonic Software, VeriSign. Web Services Policy Framework (WS-Policy), September 2004.

[12] L. Kagal. REI: A Policy Language for the Me-Centric Project. *HP Labs*. Palo Alto, USA., 2002.

[13] L. Kagal, T. Finin. Modeling Conversational Policies using Permissions and Obligations. *Journal of Autonomous Agents and Multi-Agent Systems,* 2006.

[14] Kagal, L., Finin, T., Joshi, A. A Policy Based Approach to Security on the Semantic Web, In *Proceedings of the second International Semantic Web Conference (ISWC),* Florida, USA, 2003.

[15] L. Kagal, T. Finin, A. Joshi. A Policy Language for a Pervasive Computing Environment. In *Proceedings of the IEEE 4th International Conference on Policies for Distributed Systems and Networks*. Bologna, Italy, 2003.

[16] Kagal, L., Paolucci, M., Srinivasan, N., Denker, G.,. Finin, T., Sycara, K. Authorization and Privacy for Semantic Web Services, In *Proceedings of First International Semantic Web Services Symposium, Palo Alto, USA,* 2004.

[17] R. Masuoka, M. Chopra, Y. Labrou, Z. Song, Wei-lun Chen, L. Kagal and T. Finin, Policy-based Access Control for Task Computing Using REI, In *Proceedings of the Policy Management for the Web Workshop*, Chiba, Japan, 2005.

[18] McIlraith, S., Son, T.C. Adapting Golog for Composition of Semantic Web Services. In *Proceedings of the 8th International Conference on Knowledge Representation and Reasoning (KR2002),* Toulouse, France, 2002.

[19] Medjahed, B., Bouguettaya, A., and Elmagarmid, A.K. Composing Web Services on the Semantic Web. *The VLDB Journal*, 12(4), 2003.

[20] OASIS. SAML 1.0 Specification Set:, 2002.

[21] OASIS. Web Services Business Process Execution Language (WSBPEL).

[22] Sycara, K., McGuinness, D., McIlraith, S., Paolucci, M., Finin, T., Denker, G., Kagal, L. OWL-S Technology for Representing Constraints and Capabilities of Web Services, In *Proceedings of the W3C Workshop on Constraints and Capabilities for Web Services*, CA, USA 2004.

[23] Universal Description, Discovery and Integration (UDDI). UDDI v. 3.0, UDDI Spec Technical Committee Specification, 2002.

[24] WebOnt. OWL Web Ontology Language. Web-Ontology (WebOnt) Working Group, World Wide Web Consortium (W3C). Online: http://www.w3.org/2001/sw/ WebOnt, 2003.

[25] World Wide Web Consortium (W3C). Web Services Description Language (WSDL), Version 1.2, W3C Working Draft, 2002.

[26] World Wide Web Consortium (W3C). SOAP Version 1.2 Part 1: Messaging Framework, W3C Proposed Recommendation, 2003.

[27] World Wide Web Consortium (W3C). Web Services Architecture Requirements. World Wide Web Consortium (W3C) Working Draft,. Online: www.w3.org/TR/2002/WD-wsa-reqs-20021114, 2002.

[28] Workflow Management Coalition (WfMC): www.wfmc.org