Towards Secure Execution Orders for Composite Web Services

Joachim Biskup[1], Barbara Carminati[2], Elena Ferrari[2], Frank Müller[1], Sandra Wortmann[1]
[1]Information Systems and Security, University of Dortmund, Germany
{joachim.biskup, frank.mueller, sandra.wortmann}@uni-dortmund.de
[2]University of Insubria, Varese, Italy
{barbara.carminati, elena.ferrari}@unisubria.it

Abstract

Recently, there has been a growing interest in web service composition and the related security issues. In this paper, we propose a framework for the decentralized execution of composite web services capable to ensure the correctness as well as the security of the execution. Our framework relies on a data structure, called container*, which is passed among the web services participating in the composition. The container is encrypted and authenticated in such a way to ensure the correctness of the execution flow as well as a set of relevant security requirements.*

1. Introduction

Large computing infrastructures, like the Internet, increase the capacity to share information and services across organizations. For this purpose, web services have gained popularity in both research and commercial sectors. A web service [22] is a software system designed to support interoperable application-to-application interactions over the Internet. Web services rely on a basic set of standards such as Universal Description, Discovery and Integration, Web Services Description Language, and Simple Object Access Protocol (SOAP [24]). Clearly, the prolific use of web services for critical and strategic applications gives rise to a major concern regarding the threats against security. The web services community has done significant work to address secure and reliable interoperability. The WS-* specifications define formats and protocols that are specifically tailored to, e.g., service description (WSDL [9]), messaging services (SOAP, WS-Adressing [23]), and quality of service (WS-Security [16]). WS-Security is the basic building block for securing web services. It is a family of specifications that especially supports *confidentiality* (in terms of ensuring that only the intended recipient of information is able to view it), *authenticity* (in terms of identifying the origin of a message), and *integrity* (in terms of detecting that no one has tampered with information in a message). However, these standards provide only efficient means to protect the communication among isolated web services. One of the major goals of web services is to make easier their composition to form more complex services. The workflow (or choreography) of composite web services is described in specific workflow specifications such as WS-BPEL [17]. So far a composite web service provider exposes the workflow description and it is typically responsible for the centralized execution of the overall process which is described.

The aim of this paper is to provide a decentralized control mechanism and a related supporting framework where the execution of the composite web service is not always in charge of a single provider. Rather, the proposed framework has been devised to delegate as much as possible the execution to participating web services, by at the same time ensuring the correctness of the control flow as well as the main security requirements. In particular, our framework ensures that the deployment will be carried on by: *(a)* following the control flow described in the WS-BPEL document; *(b)* effectively executing all operations of the workflow described by control structures in the WS-BPEL document; *(c)* ensuring that a web service accesses only those information strictly necessary for correctly executing the invoked operations; obtaining thus what we call a *secure execution order*. The central idea of our approach is that the messages exchanged among the web services participating in the composition are based on a particular data structure, called *container*. The main benefits of the proposed solution is that it does not require the presence of a dedicated web service that is responsible for the WS-BPEL choreography but, at the same time, ensures security requirements for the composition execution (see Section 4 for a detailed description).

Over the last years some proposals of access control models for atomic web services have been presented, see e.g. [10, 12].

There are also a few proposals to define authorization architectures and access control policies for composite web

2007 IEEE International Conference on Web Services (ICWS 2007)
0-7695-2924-0/07 $25.00 © 2007 IEEE

services, see e.g. [4, 14], whereas some approaches, such as [1, 6], deal with capability-based access control [5] for (composite) web services. Another relevant effort towards web service security is the one proposed in [13], where authors extend OWL-S [13], the emerging standard for semantic web service description, by proposing ontology for annotating input and output parameters of a web service with respect to its security characteristics (e.g., encryption and digital signature requirements). Other proposals in the field of composite web services, e.g., [2, 3, 7, 11, 25] address the problem of matching the security requirements of web services before composing them, in such a way that only web services compliant with respect to security requirements are composed. However, none of these proposals deals with the issue addressed by the current paper, that is, providing a framework for a decentralized and secure execution of composite web services.

In [15], authors present a code partitioning algorithm that is based on the idea of merging BPEL activities (given by a BPEL workflow) along (loop-independent) control flow graphs. The resulting partitions can be used for decentralized execution of the modeled BPEL workflow. [20] suggests to achieve decentralized data distribution among web services. This is done by introducing a new protocol layer (called DFDP-WS) that encapsulates control- and data-flow among web services. Similar to our approach, the proposals aim at not requiring a dedicated web service that is solely responsible for executing the composition. However, both proposals do not give technical mechanisms ensuring that the composition is securely executed according to the specified order.

The remainder of this paper is organized as follows. In Section 2 we present the underlying basics and introduce a running example we will use throughout the paper. Section 3 gives an overview of the proposed framework, whereas Section 4 introduces the encryption and authentication scheme we use for the container by giving a detailed description of the *layered container structure*. In Section 5 we describe the four main phases of the secure execution order. Finally, Section 6 concludes the paper.

2. A Simple Digital Library Example

As an example for a *business entity* consider a `digital library` (DigiLib). DigiLib deals with a huge amount of electronic documents. These documents are of different source material, from different kinds of topics, and from different countries. DigiLib provides different *web services* such as `acquiring documents`, `disseminating documents`, and `accessing documents`. Each web service is a collection of several *operations* that need to be invoked in order to execute the web service. Some operations depend on input parameters that are supplied by the web service user or by the result(s) of formerly executed web services. To ensure that only authorized users access the web service and to ensure the correct behaviour of the web service, input parameters need to fulfill certain preconditions. Preconditions can be met by supplying, e.g., simple string parameters or references to further objects. For instance, these objects can be *(user) credentials* that are supplied by users or formerly executed web services.

Typically, web services are distinguished in *atomic* and *composite* web services. An atomic web service is one that solely invokes operations that it consists of. A composite web service is one that (additionally) accesses other web services or, in particular, invokes operations of other web services. Hereby, these additional involved web services may be provided by the same business entity or may be provided by other business entities. A composite web service has a *control flow* and a *data flow* specification that contain information about how the component web services are connected and how the data flows from one component to another. Roughly speaking, a *WSDL document* [8] defines atomic web services as collections of operations together with abstract descriptions of the data being exchanged. Further, a *WS-BPEL document* [17] defines composite web services by describing interactions between business entities through web service operations. A WS-BPEL document gives the control flow defined by possible WS-BPEL *control structures* such as `sequence`, `if-then-else`, and `flow`.

Let assume that a library member wants to access a document, say `doc1`, and requests for executing web service `accessing documents` at DigiLib. DigiLib might preserve some documents that are not accessible to library members, or some documents accessible to library members may not be in the library repository. Typically, the requirements for accessing documents depend on copyright laws. There exist various aspects that influence a copyright law. For instance, the copyright may be solely claimed by the author(s) of the respective document. However, in the case of works made for hire, the employer is considered to be the author. Further, it is possible that state laws regulate copyrights or some other regulating publisher contracts exist. Thus, for deciding on particular requests, we introduce the business entity `publisher information network` (Publisher). Publisher supports information about copyright laws of different countries or publishers and further copyrights concerning the documents.

Recall the DigiLib member request for accessing `doc1`. In the easiest case, the requested document is in the public domain of the library. Then, DigiLib checks whether the requesting member is allowed to access `doc1` in terms of the copyright laws. Therefore, Publisher is requested to check appropriate copyright laws. To do so, DigiLib passes the member request together with additional member infor-

2007 IEEE International Conference on Web Services (ICWS 2007)
0-7695-2924-0/07 $25.00 © 2007 IEEE

mation (i.e., user credentials) to Publisher. In the easiest case, Publisher decides that the requesting member is authorized to fully access doc1. Thus, DigiLib lends the requesting member doc1. Assume that DigiLib does not have the requested document doc1 in stock. In this case, DigiLib needs to contact its authorized representative Rep1 and another digital library Lib2 that has lending contracts with DigiLib. Both Rep1 and Lib2 check whether doc1 is available and send the respective results to Publisher. Publisher checks appropriate copyright laws depending on the respective results and additionally passed (user) credentials. Afterwards, it decides which library is charged with lending doc1. The charged library delivers doc1 to DigiLib. Then, DigiLib lends doc1 to the requesting member. The simplified WS-BPEL specification given in Figure 1 sketches the workflow, and Figure 3 accordingly depicts a graphical representation.

```
1 <sequence>
2  <invoke partnerLink="DigiLib" operation="checkDoc" input="doc1"
3          output=Response />
4  <if> <condition> $Response == "Yes" </condition>
5       <invoke partnerLink="Publisher" operation="checkLaw" input="doc1"
6               input="DigiLib" input=userCreds output=RespPub />
7       <if> <condition> $RespPub == "Yes" </condition>
8            <invoke partnerLink="DigiLib" operation="lendDoc"
9                    input="doc1" />
10      </if>
11  <elseif>
12    <flow>
13     <sequence>
14       <invoke partnerLink="Rep1" operation="checkDocument"
15               input="doc1" output=RespRep1 />
16      <invoke partnerLink="Publisher" operation="checkLaw"
17       input="doc1" input=RespRep1 input=userCreds output=Response1/>
18     </sequence>
19     <sequence>
20        <invoke partnerLink="Lib2" operation="checkDocument"
21                input="doc1" output=RespLib2 />
22       <invoke partnerLink="Publisher" operation="checkLaw"
23        input="doc1" input=RespLib2 input=userCreds output=Response2/>
24     </sequence>
25    </flow>
26  </elseif>
27 </if>
28 <if> <condition> $Response1 == "Yes" </condition>
29      <invoke partnerLink="DigiLib" operation="retrieveDoc"> ...
30        <invoke partnerLink="Rep1" operation="lendDoc" input="doc1" />
31      </invoke>
32  <elseif> <condition> $Response2 == "Yes" </condition>
33       <invoke partnerLink="DigiLib" operation="retrieveDoc"> ...
34       <invoke partnerLink="Lib2" operation="lendDoc" input="doc1" />
35       </invoke>
36  </elseif>
37  <elseif> <throw Failure /> </elseif>
38 </if>
39</sequence>
```

Figure 1. A sample workflow specification

3. Overall framework

Following existing standards, we have to enforce the fundamental security requirements of availability, confidentiality, authenticity, and integrity. For securing composite web service executions, we adapt and refine the general requirements as follows.

(1) **Availability.** Each web service involved in the composition should be able to access the portion of the WS-BPEL document related to the activities it has to execute, thus to be able to invoke the corresponding operation(s). We refer to this portion of the WS-BPEL document as the *process view* of a web service. Consider the web service composition presented in Section 2, whose simplified WS-BPEL workflow is shown in Figure 1.

For instance, the process view of Rep1 consist of lines 14-15 and line 30. Furthermore, each web service must be able to access (user) credentials, if these are mentioned as input preconditions in its process view and are thus required to ensure the correct execution of the invoked operation(s).

(2) **Confidentiality.** Each web service gains no more than the minimum of necessary information for correctly executing the invoked operation(s).

(3) **Authenticity.** Each web service should be able to verify the authenticity and integrity of the accessed process view and (user) credentials.

(4) **Integrity of the execution order.** A web service should be able to access the authorized process view, (user) credentials, and some further input/output items only when the corresponding activity in the workflow has been triggered.

Referring to the scenario presented in Section 2, we have to ensure, for instance, that Publisher will be able to access the user credentials (see input userCreds, lines 6,17,23) only when the corresponding operation has to be invoked. In the case that all the libraries fail at the search of the document, the publisher does not have to invoke its operation, thus it should not be able to access the (user) credentials.

To satisfy these requirements, the messages exchanged among the web services participating in the composition are based on a particular data structure, called *container*. A container stores all the information needed for the deployment of the composite web service, that is, information about the WS-BPEL workflow and (user) credentials, and its encryption scheme satisfies the above given requirements. See Section 4 for a detailed description of the container structure.

In support of our approach, we assume the presence of a *Composite Web Service Provider* (CWSP) in charge of the *container generation*. More precisely, we assume that whenever the CWSP receives a request of a service composition, it specifies an appropriate WS-BPEL document, collects the needed (user) credentials, and generates the corresponding container. Then, in order to start the composition, the CWSP sends the container to the web service associated with the first control structure.

Whenever a web service participating in the composition receives a container it performs the following tasks: *(a)* it firstly extracts and decrypts its process view plus some additional control information, and the authorized (user) credentials; *(b)* it invokes the corresponding operation(s); *(c)* once all operations are terminated, it suitably modifies the

2007 IEEE International Conference on Web Services (ICWS 2007)
0-7695-2924-0/07 $25.00 © 2007 IEEE

received container; *(d)* finally, it sends the modified container to web service(s) associated with the next control structure(s).

To make a web service able to properly handling the container, it needs *extra operations* wrt those declared in its WSDL document. Obviously, requiring that each web service participating in a composition implements these extra operations is a strong assumption, since it implies a revision of the internal business logic of each web service. For this reason, we have devised an alternative solution, which does not require any modification to the existing web services. The solution is based on the presence of additional web services, called *Interpreters*, in charge of implementing the operations needed to handle a container.

Each business entity may host a unique Interpreter to be used like an interface between its web services and the other web services participating in a composition. Therefore, the CWSP makes available a copy of a *Trusted Interpreter* to all the business entities willing to participate with one of their web services, in the composition. However, we have to consider that some business entities could refuse to install an external code inside their environment. To handle this case, we provide the possibility for a business entity to implement by itself the operations of the Interpreter, by simply following the technical description in the corresponding WSDL document (available at the CWSP site). Clearly, we have to take into account that the Interpreters implemented by the business entities are potentially untrusted, that is, an *untrusted Interpreter* could potentially *modify the execution order* of the composition or *evaluate branch conditions in a wrong way*. To this purpose, as will be more clear in Section 4, we have devised an encryption scheme for the container which ensures that a WS-BPEL workflow not containing branch conditions is always processed in the correct order. Unfortunately, the devised encryption strategy is not enough to ensure the correct processing of a process view containing branch conditions, like for instance `if` and `while` control structures, because there is no means to ensure that an untrusted Interpreter does not maliciously evaluate a branch condition. Thus, if the web service in charge of evaluating a branch condition does not host a trusted Interpreter, we assume that the corresponding container is passed to the CWSP or to an external third-party entity hosting a trusted Interpreter. Then, the external trusted Interpreter will process the process view by honestly evaluating the branch condition and directly invoking the operations of the passing web service. Indeed, during container generation, if a process view contains a branch condition, the CWSP checks whether the web service in charge of this control view hosts a trusted Interpreter. If this is not the case, the CWSP generates the container such that, when the first activity of that control view will be triggered, the container will be sent to an external trusted Interpreter.

4. Container Structure

In this section we present the details of the container structure, which is the key component of our approach. As pointed out before, the aim of the container is to store information about the web service composition that has to be executed and the (user) credentials needed for its deployment, making this information available to all participants of the composition by at the same time ensuring the security requirements mentioned in Section 3.

As we will explain in Section 5, before generating a container, the CWSP retrieves all the needed information, that is, the WS-BPEL document and the (user) credentials $cred_1, \ldots, cred_m$, needed during the composition execution. In explaining the container structure, we assume that $\langle WS_1, \ldots, WS_n \rangle$ is the enumeration of occurrences of the web services participating in the workflow. Moreover, we assume that $pk_{WS_1}/sk_{WS_1}, \ldots, pk_{WS_n}/sk_{WS_n}$ are the respective public/private key pairs for asymmetric encryption and digital signatures.[1] The public/private key pair of CWSP is denoted as pk_{CWSP}/sk_{CWSP}.

Our container structure consists of three main layers, namely, the *credential layer*, the *output layer*, and the *payload layer*.

Credential layer. The aim of the credential layer is to store the (user) credentials needed during the execution of the composite web service, by at the same time *ensuring the availability and the confidentiality requirements*, see Section 3. This implies that each credential should be accessible only by the authorized web services, and that a web service should not be able to access credentials that are not required by its operations.

Our solution minimizes the dimension of this layer. In particular, we symmetrically encrypt each user credential with a different *credential key*, and store a unique copy of the encrypted credential into the credential layer. Then, to ensure both availability and confidentiality, we make available to each web service only the set of credential keys decrypting all and only the authorized credentials. Moreover, we assume that each credential is marked with an unencrypted label.[2]

Output layer. By means of the container it should be possible for a web service to store the results of its operation(s), thus to make them available to web services in the workflow. Once its operation(s) has terminated and before sending the particular container, a web service attaches the obtained results, hereafter output, directly in the output layer. Due to the availability and the confidentiality requirement,

[1] Usually, key pairs for encryption and digital signatures are separated. However, for the sake of a simple exposition, we refrain from that here.

[2] Dependent on the level of confidentiality these unique labels are either random or meaningful strings.

2007 IEEE International Conference on Web Services (ICWS 2007)
0-7695-2924-0/07 $25.00 © 2007 IEEE

we ensure that each output is accessible only by a web service that effectively needs it as input, and that a web service is not able to access output that is not required by its operations. Similar to the credential layer, we impose that each output is encrypted. More precisely, if the output is needed only by the CWSP, the *output encryption* is done with the public key of the CWSP. Otherwise, we introduce additional keys, called *output keys* to allow the current web service to encrypt its output and make only the authorized web services able to access it. Note that by analyzing the WS-BPEL document, for each output generated by a web service, the CWSP is able to determine which are the web services authorized to access it. Thus, in order to make the authorized web service able to decrypt the output, the CWSP includes the output key in the corresponding authorized portion of the container. Similar to the credentials, we assume that each output is marked with an unencrypted label. Thus, together with the output key, a web service is provided with the label of the corresponding output, and thereby enabling the web service to select and decrypt the encrypted output. Moreover, to prevent a decrypting web service from changing the output, the web service producing the output also has to sign it.

Payload layer. For each occurring web service WS_j in $\langle WS_1, \ldots, WS_n \rangle$, the payload layer stores the corresponding process view and some additional information needed to properly handle the container. All these data are organized in so-called *parcels*, such that for each occurring web service the payload layer contains a different parcel. More precisely, the CWSP processes the process view and generates a set of variables, hereafter called *control view*, to be inserted in the parcel. In particular, given a web service WS_j, the control view stored in its parcel contains the following variables: *(1)* `partnerLink`, specifying the URL of the next web service(s), that is, the URL(s) where it is expected that the container is sent after that WS_j has terminated its activity; *(2)* `operation`, giving the name of the operation to be invoked; *(3)* `input`, storing the label(s) of the credential(s) or output(s) that WS_j has to decrypt in order to generate the input message of its operation; *(4)* `output`, providing WS_j with the label(s) and the encryption key(s) that WS_j has to use for encrypting its output results; *(5)* `activity`, containing information about the control structure the WS_j is involved in (i.e., `flow`, `while`, `invoke`, etc.). In case of an `invoke` structure, `activity` stores information about the control structure context, i.e., whether the web service is involved in a while, a flow, etc. We will see that this information is needed by the proposed strategy for ensuring the integrity of the execution order. Analogous to the credentials, *ensuring the availability and the confidentiality requirement* implies that each parcel should be accessible only by the authorized web service, and a web service should not be able to access the parcels associated with other web services. Moreover, we have to satisfy *the integrity of the execution order*, that is, we have to ensure that the container will be exchanged among the web services strictly according to the workflow. In order to satisfy all these requirements we exploit encryption techniques. It is interesting to note that the naïve solution of encrypting the parcel with the public key of the corresponding web service is not enough. Indeed, it satisfies the availability and confidentiality of the parcel, but this strategy does not prevent possible attacks to the integrity of the execution order, i.e., a malicious web service can simply skip the next one. Thus, we impose that for each web service WS_{j+1} in $\langle WS_1, \ldots, WS_n \rangle$ the corresponding parcel is symmetrically encrypted with a random and unique encryption key, hereafter called *parcel key of* WS_{j+1} and denoted as $k_{WS_{j+1}}$. Moreover, we define the parcel of the preceding WS_j such that it contains the parcel key of WS_{j+1}, encrypted with the public key of WS_{j+1}, i.e. $Enc_{pk_{WS_{j+1}}}(k_{WS_{j+1}})$. This strategy requires that once WS_j has terminated its operation(s), it will provide WS_{j+1} with its encrypted parcel key. More precisely, before sending the container to WS_{j+1}, WS_j will extract $Enc_{pk_{WS_{j+1}}}(k_{WS_{j+1}})$ and insert it into the output layer. Passing the encrypted key as input ensures that web service WS_{j+1} will be able to decrypt its parcel only when it receives the encrypted key by the direct predecessor in the execution order. However, consider the case that a web service is the "joining last" involved in a flow structure. Here, ensuring the integrity of the execution order implies ensuring that the web service waits for outputs of all the preceding web services. In this case, the parcels of the preceding web services contain a share of the parcel key instead of the parcel key itself. More precisely, if there are t threads in a flow, the parcel key in each of the preceding parcels is replaced by a share of the parcel key, produced by a t-out-of-t secret sharing scheme [21].

To support the web service in finding the corresponding parcel in the container, we need to label each parcel. However, satisfying the confidentiality requirement implies hiding the order of the web services, except of the inevitable information about the preceding and next web service. Therefore, we use a random string of fixed length as a *label for each parcel*. Moreover, we shuffle the storage order of the parcels in the container to prevent gaining more information about the workflow specification than its control view.

To cope with authenticity and integrity of the control view and the credential(s), the CWSP signs the payload layer and the credential layer with its secret key, such that the receiving web service can verify CWSP's signature with the public key obtained during the location phase. To satisfy the authenticity and integrity of the respective outputs, we require that each web service suitably encrypts its output.

2007 IEEE International Conference on Web Services (ICWS 2007)
0-7695-2924-0/07 $25.00 © 2007 IEEE

Definition 4.1 (Container)
Let $\langle WS_1, \ldots, WS_n \rangle$ *be the enumeration of occurrences of the web services participating in the workflow. The corresponding container is the structure* $C = ((payload, credential)_{Sig_{sk_{CWSP}}}, output)$ *where:*

- *for each* WS_j *in* $\langle WS_1, \ldots, WS_n \rangle$*, the payload layer contains a symmetrically encrypted, different parcel,*
- *the credential layer contains symmetrically encrypted user credentials,*
- *the output layer contains input for and output of web services, either symmetrically or asymmetrically encrypted depending on the receiver.*

In order to generate a container, the CWSP first extracts the information needed to generate the parcels from the WS-BPEL document. Then, the CWSP generates a different parcel for each occurrence of a web service in the workflow. Once the parcels are generated, the CWSP forms the payload layer and appends the encrypted and labeled credentials as the credential layer. In addition, the CWSP signs the payload layer and the credential layer with its secret key. Finally, the CWSP includes the encrypted and labeled user inputs in the output layer. Figure 2 depicts the container structure and a parcel example which applies to the simple example given in Section 2.

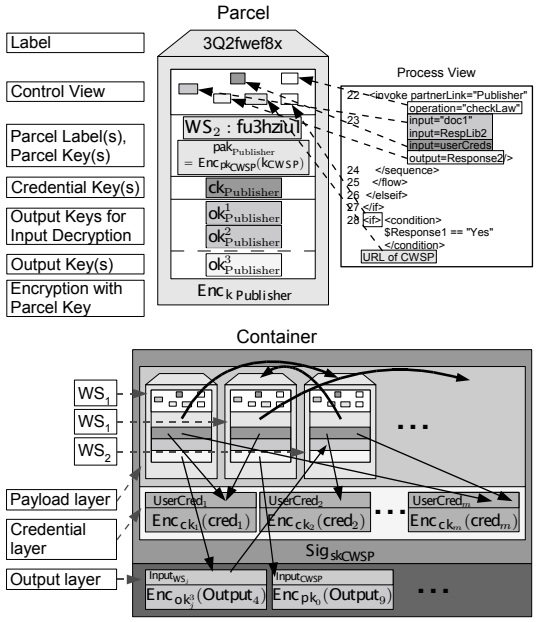

Figure 2. Structure of Parcel and Container

5. Secure Execution Order Phases

In this section we describe the four main phases of our approach, by pointing out which components of the CWSP or Interpreter are in charge of carrying on which task. Figure 4 roughly depicts the overall architecture instantiated for the example introduced in Section 2.

Offer phase. First, the CWSP retrieves a workflow WF that models the business process required to produce the requested composite service. We assume that this workflow is encoded by a WS-BPEL document, where all control structures of the workflow (e.g., `switch`, `flow`, `while`, etc.) are described. The specification of an appropriate WS-BPEL document is performed by the CWSP's `Generate WS-BPEL` component (see Figure 4). This phase may be skipped, in case that a requesting user or requesting business entity generates a workflow specification on its own. Further, we allow this workflow to be specified in other well known workflow specification languages such as e.g., [18].

Location phase. Once the WS-BPEL document has been generated, for each control structure of the selected WF the CWSP identifies a responsible web service. In this phase, the CWSP makes use of both UDDI search functionality, and semantic annotations to perform the assignment. Once a web service has been selected, the CWSP retrieves the corresponding WSDL document and extracts information about the required input messages from it. Moreover, if it is the first time that the web service participates to a CWSP composition, the CWSP contacts it in order to clarify possible ambiguities on input messages, and to make it aware that it must host the Interpreter web service. This task is performed by means of the CWSP's `LocateWS` component (see Figure 4). At the end of this phase, the CWSP knows which is the user information that is needed during the composition (i.e., the information required by input messages of web services' operations). Thus, the CSWP requires of the user his/her needed credentials.

Container generation phase. This phase, carried out by the CWSP's `GenerateCo` component (see Figure 4), is executed upon the user provided the CWSP with the required credentials. This phase is described in Section 4.

Execution phase. Once the container has been generated, the composite web service can be executed. To start the execution, the CWSP sends the generated container, an encrypted parcel key, and the label of the corresponding parcel to the Interpreter of the web service corresponding to the first control structure. To do so, the CWSP makes use of the WS-Security's key concepts: On the assumption that a container is handled as a new kind of *security token*, a SOAP message transports the corresponding signatures as *signature elements* and the parcel keys and parcel labels as *encryption elements*. Upon the Interpreter receives such a SOAP message, it firstly processes the SOAP header blocks that are targeted to it. Thereby, it decrypts the parcel key with its private key. Then, using the parcel key, it decrypts the designated parcel of the security token which includes

2007 IEEE International Conference on Web Services (ICWS 2007)
0-7695-2924-0/07 $25.00 © 2007 IEEE

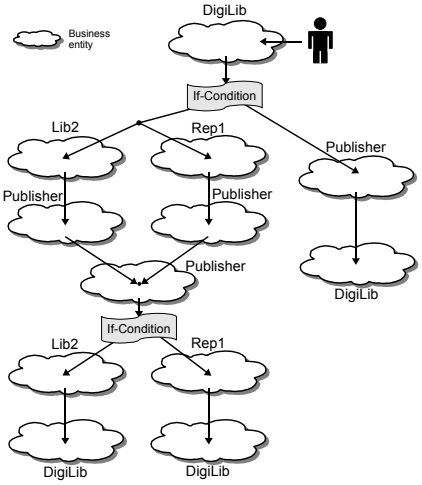

Figure 3. A sample workflow

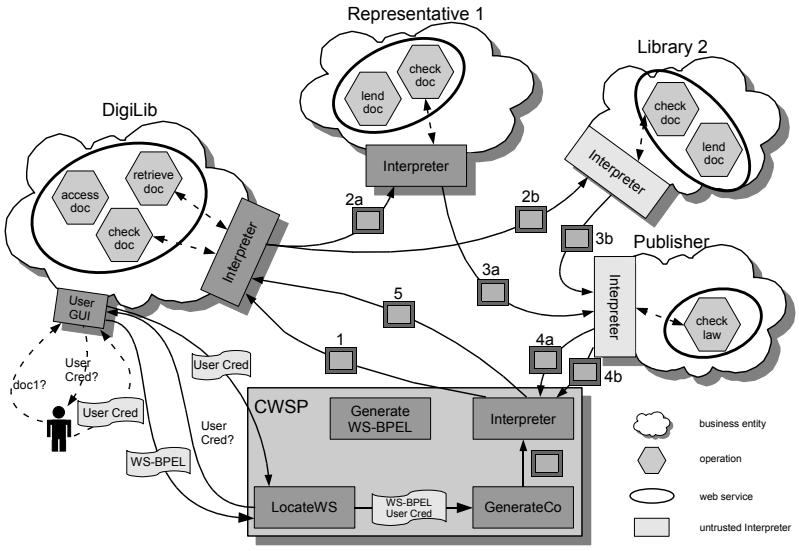

Figure 4. An overview of the architecture

the credential keys, the output keys, and the authorized control view. Moreover, it decrypts, according to the control view, the (user) credentials with the credential keys and the input(s) with the output key(s). Then, it processes the WF by properly invoking the operation(s) of the web service, which the control view refers to, with the decrypted information. Once all operations are terminated, it modifies the received security token, i.e. the container, by inserting the possible results of the invoked operation(s), encrypted with the pertinent output key(s). Finally, it extracts the information about the next web service (URL, parcel label, parcel key) and transfers the modified SOAP message to the Interpreter of the next web service.

Recall our digital library example. We assume that, by skipping the offer phase, DigiLib generates a simplified WS-BPEL document on its own and provides the CWSP with the document given in Figure 1. During the location phase, the CWSP gains the WSDL documents from the participating web services and further obtains requested (user) credentials by acquiring them from DigiLib. Then, the CWSP's component `GenerateCo` generates the corresponding container (as given in Figure 2) and the Interpreter submits it to DigiLib, see Figure 4 (step **1**). DigiLib's Interpreter extracts DigiLib's control view from the container and invokes the respective operation, i.e., `access doc`. Since we assume that DigiLib does not have the requested document `doc1` in stock, DigiLib's Interpreter modifies the container, and sends it to `Rep1` and `Lib2` (steps **2a, 2b**). The Interpreter of `Rep 1` extracts the corresponding control view, invokes `check doc`, modifies the container and sends it to Publisher (**3a**). The Publisher's Interpreter then modifies the container and invokes `check law` with the received inputs. We assume that the Interpreter of `Lib2` is implemented by the business entity itself and is thus not trusted for evaluating conditions. Because `Lib2`'s control view contains no branch conditions, the Interpreter invokes `check doc` and modifies the container. It is important to note that, even though we assume `Lib2`'s Interpreter to be untrusted, the secure execution order will be ensured. `Lib2`'s Interpreter sends the modified container to Publisher (step **3b**). The Publisher's Interpreter decrypts the container and invokes `check law` with the received inputs. Next, the results of the twice invoked `check law` needs to be evaluated, see `<if>` conditions in Figure 1 (lines 28,32). However, the Publisher's Interpreter is seen as untrusted and thus sends the modified containers to CWSP (steps **4a, 4b**). The CWSP's Interpreter evaluates the branch conditions. Then, by assuming that only the branch condition given in Figure 1, line 28 is evaluated to true, the CWSP's Interpreter modifies the respective container (i.e., received by step **4a**) and sends it to DigiLib's Interpreter (**5**). DigiLib's Interpreter invokes the web service operation `retrieve doc`. As indicated by dots in Figure 1 (lines 29,33), this operation invokes further operations, which are not considered here. We can assume that DigiLib retrieves the requested document `doc1` from `Rep1` and lends the requested document `doc1` to the requesting user.

6. Conclusion

In this paper we have presented a decentralized mechanism and a related supporting framework for ensuring a secure execution of composite web services. In particular, our framework ensures that the web service deployment is carried on by: *(a)* following the control flow described in the corresponding WS-BPEL document; *(b)* effectively execut-

2007 IEEE International Conference on Web Services (ICWS 2007)
0-7695-2924-0/07 $25.00 © 2007 IEEE

ing all operations of the workflow described by the control structures into the WS-BPEL document; *(c)* ensuring that a web service accesses only those information strictly necessary for correctly executing the invoked operations.

We are currently implementing a prototype system to test the performance of the framework with different kinds of workflows. Our intention is to have a prototype as much as possible compliant with existing web service standards. For this reason, we are basing our implementation on the WS-Security family. Especially the emerging standard WS-Context [19] is expected to be a suitable platform for realizing our framework. Roughly speaking, the WS-Context framework enables the correlation of distributed business entities that offer web services with shared operations by providing *context* information to the concerned business entities. Although the WS-Context framework does not consider entire execution orders, we believe that we could adapt the framework's mechanism, i.e., integrating context information into a SOAP message header. It is obvious that a control view can be seen as a new kind of context, and further the dedicated Context Service can be expanded with the extra operations of an Interpreter.

Besides the implementation, we plan to provide a formal analysis of the security properties achieved by the proposed approach. Moreover, we plan to extend the container structure to support dynamic composition of web services. This is the case, for instance, of a workflow requiring to invoke an operation whose details (e.g., name of an operation) are provided as output of some previous activity.

References

[1] S. Agarwal, B. Sprick, and S. Wortmann. Credential based access control for semantic web services. In T. Payne, editor, *AAAI Spring Symposium - Semantic Web Services*, pages 44 – 52, 2004.

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

[3] E. Bertino, A. Squicciarini, and D. Mevi. A fine-grained access control model for web services. In *Proceedings of the 2004 IEEE International Conference on Services Computing*, pages 33–40, Shangai, China, 2004.

[4] R. Bhatti, E. Bertino, and A. Ghafoor. A trust-based context-aware access control model for web-services. In B. Werner, editor, *IEEE International Conference on Web Services (ICWS'04)*, pages 184 – 191, San Diego, CA, USA, 2004. IEEE Computer Society Press.

[5] J. Biskup and Y. Karabulut. A hybrid PKI model: Application to secure mediation. In E. Gudes and S. Shenoi, editors, *Proceedings of the 16th Annual IFIP WG 11.3 Working Conference on Data and Application Security*, pages 271 – 282. Kluwer Academic Publishers, 2003.

[6] P. A. Bonatti and P. Samarati. A uniform framework for regulating service access and information release on the web. *Journal of Computer Security*, 10(3):241 – 271, 2002.

[7] B. Carminati, E. Ferrari, and P. Hung. Secure conscious web service composition. In *Proceedings of the IEEE International Conference on Web Services*, Chicago, USA, 2006.

[8] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana. Web services description language (WSDL) version 2.0. http://www.w3.org/TR/wsdl20, 2006.

[9] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, 2001.

[10] M. Coetzee and J. Eloff. Towards web service access control. *Computers & Security*, 23(7):559 – 570, 2004.

[11] S. Haibo and H. Fan. A context-aware role-based access control model for web services. In *Proceedings of the 2004 IEEE International Conference on on e-Business Engineering*, 2005.

[12] J. B. Joshi, W. G. Aref, A. Ghafoor, and E. H. Spafford. Security models for web-based applications. *Communications of the ACM*, 44(2):38 – 44, 2001.

[13] L. Kagal, M. Paolucci, N. Srinivasan, G. Denker, T. Finin, and K. Sycara. Authorization and privacy for semantic web services. In *Proceedings of the AAAI Spring Symposium, Workshop on Semantic Web Services*, 2005.

[14] H. Koshutanski and F. Massacci. Interactive access control for web services. In Y. Deswarte, F. Cuppens, S. Jajodia, and L. Wang, editors, *Proceedings of the 19th IFIP International Information Security Conference (SEC'04)*, pages 151 –166, Toulouse, France, 2004. Kluwer Academic Publishers.

[15] M. G. Nanda, S. Chandra, and V. Sarkar. Decentralizing execution of composite web services. In *OOPSLA*, pages 170–187, 2004.

[16] OASIS. Web Services Security. http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf, 2004. OASIS Standard.

[17] OASIS. Web Services Business Process Execution Language (WS-BPEL), version 2.0. http://www.oasis-open.org/apps/org/workgroup/wsbpel, 2005. OASIS Standard.

[18] OASIS. ebXML BPSS (ebBP), version 2.0.4. http://docs.oasis-open.org/ebxml-bp/2.0.4/OS, 2006. OASIS Standard.

[19] OASIS. Web Services Context Specification (WS-Context) version 1.0. http://docs.oasis-open.org/ws-caf/ws-context/v1.0/OS/wsctx.html, 2007. OASIS Standard.

[20] L. M. Patcas, J. Murphy, and G.-M. Muntean. Middleware support for data-flow distribution in web services composition. In *ECOOP2005 PhDOOS Workshop and Doctoral Symposium*, 2005.

[21] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, 1979.

[22] World Wide Web Consortium (W3C). Web services activity. http://www.w3.org/2002/ws, 2002 – 2005.

[23] W3C. Web Services Addressing. http://www.w3.org/Submission/ws-addressing/, 2004. W3C Member Submission.

[24] W3C. SOAP, version 1.2. http://www.w3.org/TR/soap12-part1/, 2007. W3C Recommendation.

[25] E. Yuan and J. Tong. Attribute based access control (ABAC) for web services. In *Proceedings of the IEEE International Conference on Web Services*, Orlando, Florida, 2005.

2007 IEEE International Conference on Web Services (ICWS 2007)
0-7695-2924-0/07 $25.00 © 2007 IEEE