Verification of Access Control Requirements in Web Services Choreography

Federica Paci[1]   Mourad Ouzzani [2]   Massimo Mecella[3]

[1]Computer Science Department, Purdue University
paci@cs.purdue.edu
[2]Cyber Center, Discovery Park, Purdue University
mourad@cs.purdue.edu
[3] Dipartimento di Informatica e Sistemistica, SAPIENZA - Università di Roma
mecella@dis.uniroma1.it

Abstract

Web services choreography is used to design peer-to-peer applications where each peer is potentially a Web service. It defines the required behavior of participating Web services along with their interactions through message exchanges. Implementing a complex system described by a choreography requires selecting actual Web services whose individual behaviors are compatible with the overall behavior described by the choreography. Although the selected Web services implement the specified behavior, they may not be able to interact due to the policies they enforce to protect their resources. A Web service'resource can be an operation or a credential type to be submitted to be able to invoke an operation. In this paper, we propose a novel approach to determine at design time whether a choreography can be implemented by a set of Web services based on their access control policies and the disclosure policies regulating the release of their credentials. We model both Web services and Web services choreography as transition systems and represent Web services credential disclosure policies as directed graphs. We then verify that all possible conversations of the Web services choreography can be implemented by matching credential disclosure policies of the invoker Web service with the access control policy of the Web services being invoked. We propose a resource release graph to enable this verification.

1   Introduction

Web services choreography has been introduced to standardize the design phase of complex peer-to-peer applications in which each peer can be implemented by a Web service. A Web services choreography specifies the behavior that each peer must have along with the interactions between the peers. Any Web service that would like to join the choreography would need to conform to that specification.

Several research efforts have been devoted to develop formalizations for Web services choreography and approaches for verifying that a choreography implementation is compliant with the choreography specification [6, 4, 7]. Most of these efforts thus focus on the issue of determining whether the behavior of the Web services implementing a choreography matches the one described by the choreography specification. However, verifying that a Web services choreography can be implemented by a set of Web services is not only a matter of behavior conformance; before being able to interact with each other, Web services need to establish a mutual level of trust.

Trust can be established using various strategies, such as through the use of credentials or reputation systems. In this paper, we focus on the use of credentials for trust establishment among Web services participating in a choreography. In Web service-based distributed applications where invokers are not known a priori, access control decisions are usually made based on the attributes of the entity requesting access to a particular resource, rather than his or her identity. We assume that each Web service is characterized by credentials certifying that it has certain properties [9]. Moreover, we assume that the invocation of each Web service operation is controlled by access control policies; such policies establish which credentials the invoker Web service must possess in order to be able to invoke the operation. It is also important to notice that some of those credentials may contain sensitive information about the Web services to which they refer. Therefore, disclosure of such credentials must be regulated by *credential disclosure policies*. It is thus clear that verifying whether a choreography can be implemented requires determining whether the invoker Web services can disclose the credentials specified in the access control policies of the Web service providing the invoked operation.
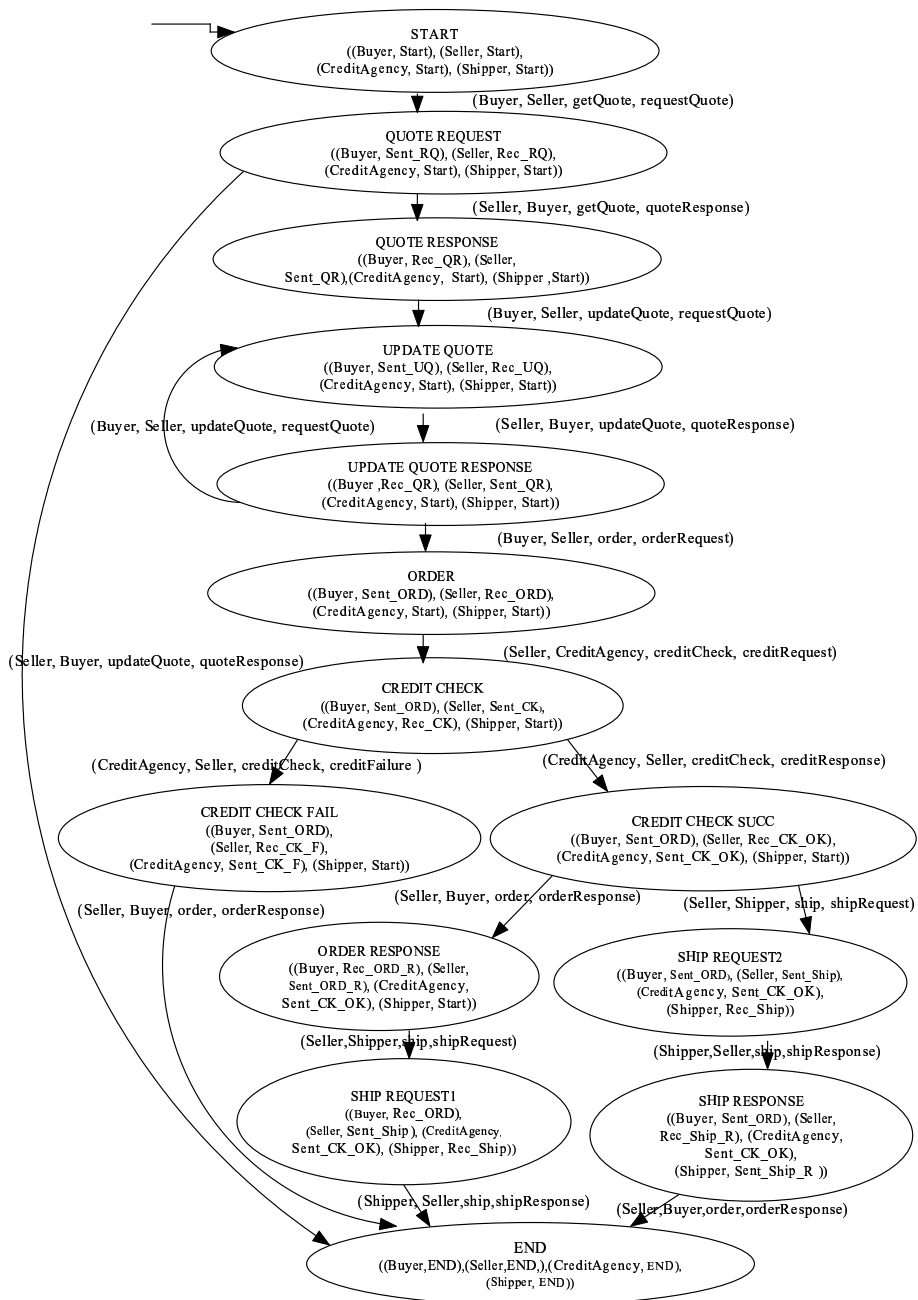
START
((Buyer, Start), (Seller, Start),
(CreditAgency, Start), (Shipper, Start))

(Buyer, Seller, getQuote, requestQuote)

QUOTE REQUEST
((Buyer, Sent_RQ), (Seller, Rec_RQ),
(CreditAgency, Start), (Shipper, Start))

(Seller, Buyer, getQuote, quoteResponse)

QUOTE RESPONSE
((Buyer, Rec_QR), (Seller,
Sent_QR),(CreditAgency, Start), (Shipper ,Start))

(Buyer, Seller, updateQuote, requestQuote)

UPDATE QUOTE
((Buyer, Sent_UQ), (Seller, Rec_UQ),
(CreditAgency, Start), (Shipper, Start))

(Buyer, Seller, updateQuote, requestQuote)

(Seller, Buyer, updateQuote, quoteResponse)

UPDATE QUOTE RESPONSE
((Buyer ,Rec_QR), (Seller, Sent_QR),
(CreditAgency, Start), (Shipper, Start))

(Buyer, Seller, order, orderRequest)

ORDER
((Buyer, Sent_ORD), (Seller, Rec_ORD),
(CreditAgency, Start), (Shipper, Start))

(Seller, CreditAgency, creditCheck, creditRequest)

CREDIT CHECK
((Buyer, Sent_ORD), (Seller, Sent_CK),
(CreditAgency, Rec_CK), (Shipper, Start))

(CreditAgency, Seller, creditCheck, creditFailure)

(CreditAgency, Seller, creditCheck, creditResponse)

(Seller, Buyer, updateQuote, quoteResponse)

CREDIT CHECK FAIL
((Buyer, Sent_ORD),
(Seller, Rec_CK_F),
(CreditAgency, Sent_CK_F), (Shipper, Start))

CREDIT CHECK SUCC
((Buyer, Sent_ORD), (Seller, Rec_CK_OK),
(CreditAgency, Sent_CK_OK), (Shipper, Start))

(Seller, Buyer, order, orderResponse)

(Seller, Buyer, order, orderResponse)

(Seller, Shipper, ship, shipRequest)

ORDER RESPONSE
((Buyer, Rec_ORD_R), (Seller,
Sent_ORD_R), (CreditAgency,
Sent_CK_OK), (Shipper, Start))

SHIP REQUEST2
((Buyer, Sent_ORD), (Seller, Sent_Ship),
(CreditAgency, Sent_CK_OK),
(Shipper, Rec_Ship))

(Seller,Shipper,ship,shipRequest)

(Shipper,Seller,ship,shipResponse)

SHIP REQUEST1
((Buyer, Rec_ORD),
(Seller, Sent_Ship), (CreditAgency,
Sent_CK_OK), (Shipper, Rec_Ship))

SHIP RESPONSE
((Buyer, Sent_ORD), (Seller,
Rec_Ship_R), (CreditAgency,
Sent_CK_OK),
(Shipper, Sent_Ship_R))

(Shipper,Seller,ship,shipResponse)

(Seller,Buyer,order,orderResponse)

END
((Buyer,END),(Seller,END,),(CreditAgency, END),
(Shipper, END))

Figure 1. Transition System of Bartering Choreography

In this paper, we propose a novel approach to decide, at design time, if a choreography can be implemented by a set of Web services by verifying that their access control policies and credential disclosure policies match. We represent the behavior of both the Web services choreography and the different candidate Web services by a non deterministic transition system. The access control policies of the candidate Web services are modeled as transition preconditions. We assume that each Web service publishes, in addition to its behavior, the credentials that it is willing to disclose and the disclosure policies regulating the release of these credentials. The disclosure policies are represented as directed graphs. Our approach verifies then that all possible conversations of the Web services choreography can be implemented by a given set of Web services based on their access control and disclosure policies. We perform this verification by checking that the message exchanges composing the conversations can be implemented. More specifically, we represent each message exchange by its resource release graph which, through a labeling process, can determine the implementability of that message exchange. The resource release graph is a directed graph where the root node represents the operation that causes the message exchange and the child nodes are the credentials specified in the access control policy of the Web service providing the operation. To each of these child nodes, we recursively append the corresponding disclosure policies of the provider and invoker Web services. Each node is then labeled as deliverable according to its underlying disclosure policies. If the root node is deliverable, the message exchange can be implemented. For the message exchanges that are not realizable, i.e., their root nodes are labeled as undeliverables, we suggest how to modify the access control and disclosure policies to make them realzaible.

The paper is organized as follows. Section 2 introduces transition system modeling for a Web services choreography. In Subsection 3, we describe how to represent the behavior of a Web service, and its access control and disclosure policies. In Section 4, we introduce our verification approach based on the resource release graph. Section 5 presents the related works. Section 6 concludes the paper.

2 Modeling Web Services Choreography

In our model, a Web services choreography is based on the notion of *roles* and *message exchanges* between roles. A role defines the behavior a Web service must exhibit to participate in the choreography. A message exchange represents the realization of a collaboration between two roles and the means by which a choreography can evolve. Each message exchange is associated with an operation offered by a role and implies an exchange of information between the invoker role and the role providing the operation. We

model a Web Service choreography as a non deterministic transition system. A sequence of message exchanges is called *conversation*.

Definition 1 (WS-Choreography Transition System)
A choreography of roles $r_1, r_2, ..., r_n$ is represented by a non deterministic transition system $TS_{choreo} = (S, A, T, s_o, s_f)$. S is a set of choreography states. Each state is a tuple of the form $((r_1, state_1), (r_2, state_2), ..., (r_n, state_n))$ where in each tuple $(r_i, state_i)$, $state_i$ represents the state of role r_i. $s_0 \in S$ is the initial state and $s_f \in S$ is the final state. A is a set of message exchanges. Each message exchange is represented by a tuple (r_s, r_d, μ, m), where r_s is the role sender of the message, r_d is the role receiver of the message, μ is an r_d's operation invoked by r_s that triggers the message exchange, and m is the type of the message sent. $T \subseteq S \times A \times S$ is the transition relation. A transition $(s, a, s') \in T$ if $a = (r_s, r_d, \mu, m)$ and the tuples $(r_s, state_s)$ and $(r_d, state_d)$ in state s are replaced by the tuples $(r_s, state'_s)$ and $(r_d, state'_d)$ in state s' (respectively) due to the invocation of the operation μ.

In this work, we focus on message exchanges where a role sends a message to invoke another role's operation. Indeed, only the invocation of an operation is restricted by an access control policy; if an operation can be invoked, then the sending of the reply message to the invoker can be performed as well.

Example 1 *Figure 1 is an example of Web services choreography describing a bartering process [1]. Each state in the transition system is labeled with the state in which each role is; each transition is labeled with the name of the role that sends a message, the name of the receiver role, the name of the operation that triggers the message exchange and the type of the message sent. The choreography involves four roles:* Buyer, Seller, CreditAgency *and* Shipper. *The* Buyer *requests a good's price from the* Seller *and the* Seller *responds with the price. The* Buyer *can then either decide not to progress or to request a price update to the* Seller *which replies with the updated price. The* Buyer *can keep asking for a price update or decide to place the order. Once the* Seller *receives the order from the* Buyer*, it contacts the* CreditAgency *to check the* Buyer*'s credit. The* CreditAgency *can confirm the credit or not. If the the credit is confirmed, then the* Seller *contacts both the* Shipper *to arrange the shipment of the good and the* Buyer *to inform it about the completion of the order. If the* Buyer*'s credit is not sufficient, then the* Seller *informs the* Buyer *that the*
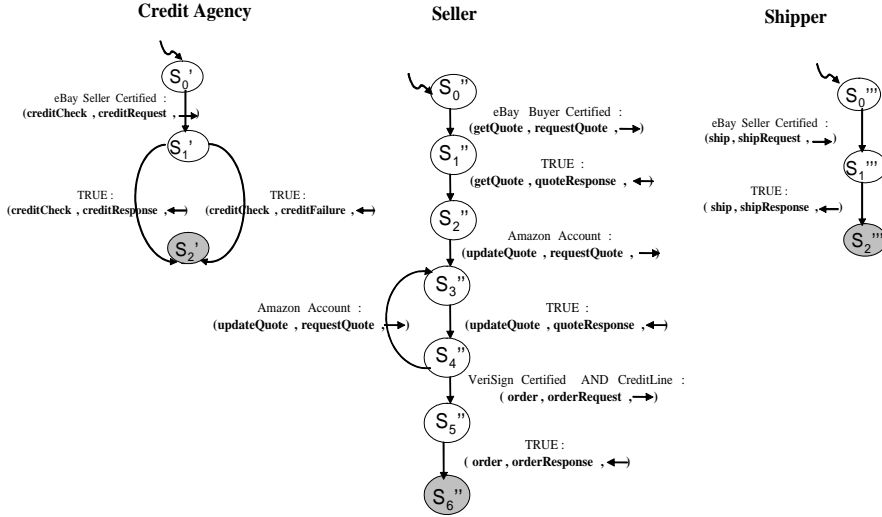

Figure 2. Transition systems of Possible Web services for Roles `CreditAgency,` `Seller,` **and** `Shipper`

order cannot go through. The following is an example of a conversation in which the `Buyer`*'s order cannot be completed because the* `Buyer`*'s credit is not sufficient:*

```
(Buyer,Seller, getQuote, requestQuote)    •(Seller,
Buyer, getQuote, quoteResponse)    •(Buyer, Seller,
updateQuote, requestQuote)          •(Seller, Buyer,
updateQuote, quoteResponse)         •(Buyer, Seller,
order, orderRequest)       •(Seller, CreditAgency,
creditCheck, creditRequest)            •(CreditAgency,
Seller, creditCheck, creditFailure)          •(Seller,
Buyer, order, orderResponse).
```

3 Access Control and Credentials

In this section, we first present how a Web service's behavior and the access control policies for its operations can be modeled. Then, we illustrate how a Web service specifies disclosure policies for its credentials.

3.1 Web Services Access Control Requirements

The behavior of a Web service describes the set of operations it exports and constraints on the possible conversations it can execute. The Web service operations can be invoked only by clients that own the credential types specified in the access control policies associated with the operations. We represent the behavior of a Web service and its access control policies as a non deterministic finite transition system. The transition system describes the messages that are received and sent by a Web service as effect of the execution

of its operations.

Definition 2 (Web service transition system) *The transition system of a Web service* WS *is a tuple* $TS_{WS} = (\Sigma, S, Prec, \delta, s_0, s_f)$. Σ *is the alphabet of the transition system. Each element in* Σ *is a tuple* (op, msg, dir) *where* op *represents a* WS*'s operation,* msg *represents an input or output message of* op *and* dir *is equal to* $\rightarrow$ *if* msg *is an input message or to* $\leftarrow$ *if it is an output message.* S *is a finite set of states.* $s_0 \in S$ *is the initial state, and* s_f *is the final state.* $Prec$ *is the set of transition preconditions.* $\delta : S \times \Sigma \times Prec \rightarrow S$ *is the transition relation.*

A state in the transition system represents the state of the interaction between a client and the Web service [8]. A transition represents the receipt of an input message or the sending of an output message of an operation. Access control policies are represented as preconditions that must be satisfied for a transition to be fired. The precondition for a transition representing the receipt of an operation's invocation message is a conjunction or a disjunction of credential types specified in the operation access control policy. The precondition of a transition corresponding to the sending of an output message is always true since we only need to verify that the message to invoke an operation can be sent.

Example 2 *In Figure 2, we represent the transition systems of three Web services that may be selected to play the roles* `Seller,` `CreditAgency,` *and* `Shipper` *introduced in Example 1. The* `Shipper` *service receives the invocation message* `shipRequest` *of* `ship` *operation and replies with a* `shipResponse` *message. A client that wants to*
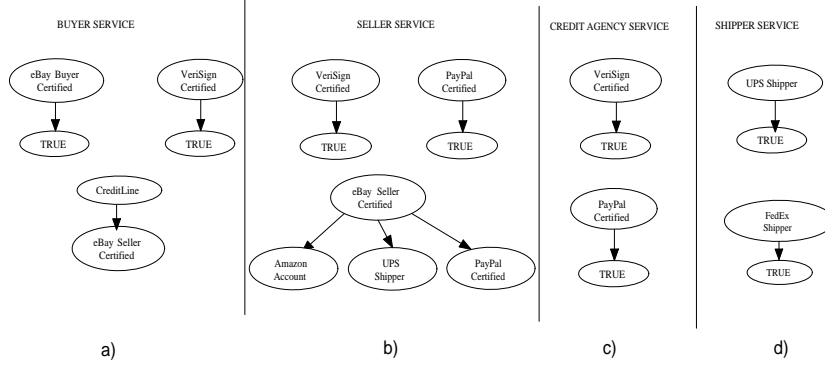
Figure 3. Disclosure Policy Graphs of Web services implementing `Buyer`, `Seller`, `CreditAgency`, **and** `Shipper` **Roles**

send the invocation message `shipRequest` *must prove to have a credential of type* `ebay Seller Certified`.

3.2 Web Services Credential Disclosure Policies

Credentials should be carefully disclosed during an interaction. For the purpose of our work, we classify credentials as being *sensitive* or *non sensitive*. Sensitive credentials can be only released after trust has been established while non sensitive credentials can be released without restriction. The submission of sensitive credentials is regulated by disclosure policies defined as follows.

Definition 3 (Disclosure policies) *A* disclosure policy *regulating the disclosure of a credential C is an expression of the form $C \leftarrow credexpr$ where C is the credential type whose release is protected by the policy and $credexpr$ is either a conjunction $C_1 \wedge C_2 \wedge ... \wedge C_n$ of credential types, a disjunction $C_1 \vee C_2 \vee ... \vee C_n$ of credential types, or the boolean value $TRUE$.*

If a disclosure policy is of the form $C \leftarrow C_1 \wedge C_2 \wedge ... \wedge C_n$ (resp. $C \leftarrow C_1 \vee C_2 \vee ... \vee C_n$), then the policy is satisfied only if the consumer of credential C proves to own the credentials of types $C_1, C_2, ...,$ and C_n (resp. at least one credential type from $C_1, C_2, ...,$ and C_n). If the disclosure policy is of the form $C \leftarrow TRUE$, then credential C can be released without any restriction. We represent disclosure policies as directed graphs called *disclosure policy graphs*. Nodes in such graphs can be circular nodes, representing credential types, or rectangular nodes, representing the conjunction of credential types.

The disclosure policy graph of $C \leftarrow C_1 \wedge C_2 \wedge ... \wedge C_n$ has a circular root node labeled with C and a rectangular child node having circular child nodes labeled with $C_1, C_2, ...,$

C_n. The disclosure policy graph of $C \leftarrow C_1 \vee C_2 \vee ... \vee C_n$ is a graph with a circular node labeled with C having circular child nodes labeled with $C_1, C_2, ..., C_n$. The disclosure policy graph of a disclosure policy $C \leftarrow TRUE$ has a circular root node labeled with C and a circular child node labeled with $TRUE$.

Example 3 *Figures 3(a, b, c, & d) represents the disclosure policy graphs of the Web services implementing* `Buyer`, `Seller`, `CreditAgency`, *and* `Shipper` *roles respectively. For example, the* `Buyer` *Web service is willing to disclose the credentials* `eBay Buyer Certified` *and* `VeriSign Certified` *without any restriction. However,* `CreditLine` *can be released only if the credential consumer provides credentials* `eBay Seller Certified`.

4 Web services Access Control Requirements Verification

In this section, we present our approach to determine whether a choreography can be implemented by a set of Web services by analyzing the access control policies they enforce and the credentials that they are willing to disclose. We assume that an application analyst has already provided the transition system representing the choreography and selected a set of Web services whose behaviors match the ones of the roles in the choreography [7, 5, 4]. The descriptions of the Web services specify their behavior, the access control policies protecting their operations, and disclosure policies regulating the release of their credentials.

Although the selected Web services implement the behavior specified in the choreography, they may not be able to perform the described conversations due to the access
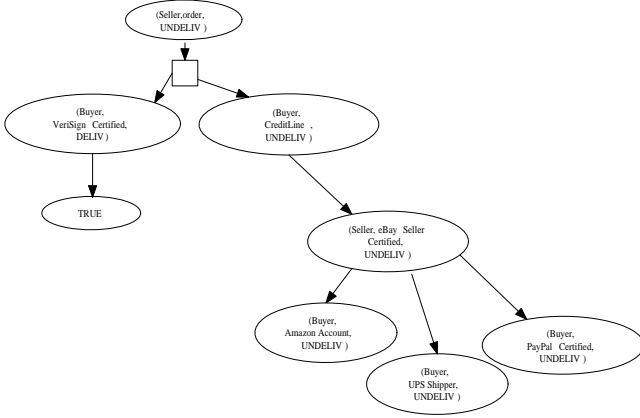
Figure 4. Resource Release Graph of $(Buyer, Seller, order, orderRequest)$ **Message Exchange**

control policies they enforce. The verification process consists in checking that all possible conversations going from the initial state to the final state in the choreography transition system can be implemented according to the operation access control and disclosure polices of the selected Web services. The main step of the verification process is to check that each message exchange (r_s, r_d, μ, m) in the choreography transition system can be *implemented* or *realized*. The conversations composed of message exchanges that are realizable are, in turn, realizable. A message exchange (r_s, r_d, μ, m) is realizable if the disclosure policies of the Web service implementing role r_s allow the release of credential types specified in μ's access control policy. The verification of whether the message exchange (r_s, r_d, μ, m) can be implemented is performed by analyzing the *resource release graph*. In the following subsections, we first show how this graph is generated and then introduce the verification algorithm.

4.1 Resource Release Graph

The resource release graph of a message exchange (r_s, r_d, μ, m) is generated based on μ's access control policy and the disclosure policies of the Web services implementing roles r_s and r_d.

Definition 4 (Resource Release Graph) *The resource release graph of a message exchange (r_s, r_d, μ, m) is a directed graph (V, E). The set of nodes V contains two different types of nodes: (i) circular nodes model credential types specified in μ's access control policy or in the disclosure policies of Web services implementing roles r_s and r_d and (ii) rectangular nodes model the conjunction of its circular child nodes. The root of the graph is a circular node rep-*

resenting the operation μ that triggers the sending of message m. Each circular node in the graph is labeled with the tuple $(ResourceOwner, ResourceName, State)$ where ResourceOwner is equal to r_s or r_d, ResourceName is equal to μ or to a credential type name, and State indicates whether the resource associated with the node is deliverable ($DELIV$), not deliverable ($UNDELIV$), or not set yet ($NULL$). E is the set of arcs connecting circular or rectangular nodes.

If the root of the resource release graph is labeled with $DELIV$, the message exchange is realizable. The creation of the resource release graph of a message exchange (r_s, r_d, μ, m) is a two-phase process. In the first phase, the graph is created based on μ's access control policy and the disclosure policies of the Web services implementing roles r_s and r_d. In the secnd phase, the graph is traversed from the root to the leaves to mark the nodes as deliverable or not based on the disclosure policies of Web services implementing roles r_s and r_d.

The first phase of graph generation consists of the following steps: (1) A circular node representing the root node and labeled with $(r_d, \mu, NULL)$ is created. (2) If the precondition of μ is a conjunction of credential types $C_1 \wedge C_2 \wedge ... \wedge C_n$, a rectangular node along with its child circular nodes labeled with $(r_s, C_1, NULL)$, $(r_s, C_2, NULL)$, ..., $(r_s, C_n, NULL)$ are created. An arc links the root node to the rectangular node. If the precondition is a disjunction of credential types $C_1 \vee C_2 \vee ... \vee C_n$, circular nodes labeled with $(r_s, C_1, NULL)$, $(r_s, C_2, NULL)$, ..., $(r_s, C_n, NULL)$ are added as child nodes to the root. (3) If the Web service implementing role r_s owns credential types C_1, C_2, ..., C_n and the release of these credential types is ruled by disclosure policies, then each node labeled with $(r_s, C_i, NULL)$ is linked with its disclosure policy graph. Each circular node in the disclosure policy graph is labeled with r_d, the name of a credential type, and with $NULL$. If the Web service implementing r_s does not own any of the credential types C_i, no child node is added to $(r_s, C_i, NULL)$ circular node. (4) If the Web service implementing role r_d has disclosure policies associated with credential types specified in C_1, C_2, ..., C_n's disclosure policies, we repeat step 3 using r_d's disclosure policies. The two steps 3 and 4 are repeated till the leaves of the graph are all circular nodes labeled with $TRUE$ or with the name of a credential type which cannot be provided by the Web services implementing r_s or r_d.

Once built, we traverse the resource graph (second phase) to label each circular node as deliverable or undeliverable. We apply the following labeling rules: (1) a node having a rectangular child node is labeled with $DELIV$ only if all its child nodes are labeled with $DELIV$; (2) a node having circular child nodes is labeled with $DELIV$ if at least one of its child nodes is labeled with $DELIV$;

(3) a node having a circular child node labeled with $TRUE$ is labeled with $DELIV$; (4) a leaf node not labeled with $TRUE$ is labeled with $UNDELIV$.

4.2 Verification Protocol

The Verification algorithm has as inputs TS_{Choreo}, the choreography transition system, the precondition tables $Prec\_Table_1, Prec\_Table_2...Prec\_Table_n$ and the disclosure policies $DiscPolSet_1$, $DiscPolSet_2, ..., DiscPolSet_n$ of the implementing Web services. It returns the set of conversations $Auth\_Conv$ that can be performed by the selected Web services. If some of the conversations cannot be performed, it returns the set $PolMod$ of access control and disclosure policies that need to be modified to implement the conversations.

The procedure computeConversations (TS) computes the set $Conv$ of all possible conversations in TS using the approach we proposed in [8] (line 1). Then, for each conversation $conv_i$ in $Conv$, the set Me_i of all message exchanges (r_s, r_d, μ, m) composing the conversation is created (lines 2-3). Me_i contains only the message exchanges where a Web service sends the operation's invocation message to the Web service that provides it. Then, the set Me of all the Me_i is built (line 4). For each message exchange (r_s, r_d, μ, m) in Me, the resource release graph is built by procedure buildResourceReleaseGraph (line 7). Then, procedure LabelNodes labels the nodes of the resource release graph with $DELIV$ or $UNDELIV$ and returns the state of the root node (line 8). If the state of the root is deliverable, the message exchange (r_s, r_d, μ, m) associated with the invocation of operation μ can be performed. (r_s, r_d, μ, m) is added to the set of authorized message exchanges $AuthMe$ (lines 9-10).

Once $AuthMe$ is built and to determine if conversations $conv_i$ in $Conv$ can be implemented, we check that Me_i is a subset of $AuthMe$ (line 14). If this is the case, all the message exchanges in $conv_i$ are realizable. Hence, $conv_i$ is added to the set $Auth\_Conv$ of conversations that can be implemented (line 15). Otherwise, we compute set $NoAuthMe_i$, the complement set of Me_i in $AuthMe$ (line 17). We are thus able to determine which are the parts of the conversation $conv_i$ that cannot be implemented. Then, for each message exchange (r_s, r_d, μ, m) in $NoAuthMe_i$, procedure computePathWithUndelivNodes determines the paths in the resource release graph where circular nodes are marked as $UNDELIV$ (line 19). By determining to which Web service the undeliverable credential types belongs to, we can suggest ways to modify the access control policies or the disclosure policies of the interacting Web services to enable the message exchange.

Example 4 *We want to check if the Web services*

Algorithm 1: Access Control Requirements Verification

Require: TS_{Choreo} the transition system of a choreography of n roles, the preconditions tables $Prec\_Table_1, Prec\_Table_2...Prec\_Table_n$[2] and the disclosure policies $DiscPolSet_1, DiscPolSet_2, ..., DiscPolSet_n$ of Web services implementing role r_i $i= 1, ..., n$

1: $Conv$:=computeConversations(TS);
2: **for** $conv_i \in Conv$ **do**
3:     Me_i:=$\{(r_s, r_d, \mu, m) \in conv_i : m$ is μ's operation invocation message $\}$;
4: **end for**
5: Me:= $\{\bigcup Me_i\}$;
6: **for** $(r_s, r_d, \mu, m) \in Me$ **do**
7:     RG:=buildResourceReleaseGraph$((r_s, r_d, \mu, m))$;
8:     $Root\_Status$:=LabelNodes(RG);
9:     **if** $Root\_Status == DELIV$ **then**
10:        $Auth\_Me$.add$((r_s, r_d, \mu, m))$;
11:     **end if**
12: **end for**
13: **for** $conv_i \in Conv$ **do**
14:     **if** $Me\_i \subseteq Auth\_Me$ **then**
15:        $Auth\_Conv$.add$(conv_i)$;
16:     **else**
17:        $NoAuthMe_i$:= Me_i - $Auth\_Me$;
18:        **for** $(r_s, r_d, \mu, m) \in NoAuthMe_i$ **do**
19:           $PolMod$:=computePathWithUndelivNodes$((r_s, r_d, \mu, m)$;
20:        **end for**
21:     **end if**
22: **end for**
23: **return** $(Auth\_Conv, PolMod)$;

choreography illustrated in Figure 1 can be implemented by the Web services in Figure 2. The set $Conv$ contains seven different conversations. We focus on the conversation reported in Example 1. Me_1=$\{$(Buyer,Seller, getQuote, requestQuote), (Buyer,Seller, updateQuote, requestQuote), (Buyer,Seller, order, orderRequest), (Seller, CreditAgency, creditCheck, creditRequest)$\}$ is the set of invocation message exchanges associated with this conversation. The conversation is not realizable because the message exchange (Buyer, Seller, order, orderRequest) is not realizable. The root node in the resource release graph of (Buyer,Seller, order, orderRequest), reported in Figure 4, is labeled with $UNDELIV$. The root is not deliverable because all the credential types in the subgraph, having CreditLine as root node, are labeled with $UNDELIV$. To make the root node deliverable, we have several options: we can modify the disclosure policy of eBay Seller Certified substituting one of the credentials types Amazon Account, UPS Shipper and PayPal Certified with one of the credential types of the Buyer Web service like eBay Buyer Certified or we can make eBay Seller Certified immediately deliverable. Another option is to modify the access control policy of the operation order substituting the conjunction of credential types VeriSign Certified and CreditLine with a disjunction.

5 Related Work

Most previous research on Web services choreography dealt with choreography formalization and verification of choreographies' properties. Web services Choreography Description Language (WS-CDL) [6] is an XML-based language that gives a global view of the interaction rules among Web services collaborating to reach a common business goal. The works of Busi et al. [4] and Pistore et al. [7] propose two formal calculi to model Web services choreography and Web services orchestration. They also investigate the interdependencies between choreography and orchestration and propose a bisimulation-like notion of conformance between choreography and orchestration of Web services. Foster et al. [5] propose a formalization of Web services composition and Web services choreographies based on a finite state process (FSP) algebra. Moreover, they propose techniques to verify the compatibility among interacting compositions of Web services, the conformance between a choreography and its implementation, deadlock absence, and safety and progress properties.

The work of Robinson et al. [10] is the only proposal that investigates the problem of how to enforce access control in Web services choreographies. They propose a mechanism to derive access control policies to be enforced by each Web service covering a choreography role and an architecture to enforce such policies at runtime. Access control policies enforcement is enabled and disabled in a just-in-time manner that matches the control flow described in the choreography. The work of Robinson et al. has a different focus from our work. In our work, we focus on design-time verification before a Web services choreography is actually deployed. Our approach checks that the interactions, defined in the choreography, can take place according to Web service providers' access control policies and Web service consumers capabilities. Robinson et al., instead, deal with the automatic derivation of access control policies for the Web services participating to a choreography and with their enforcement. In fact, the two approaches can complement each other.

6 Conclusions

In this paper, we presented an approach to verify that a choreography can be implemented by a set of Web services based on their access control policies and the credentials that they are willing to disclose. We modeled both Web services and Web services choreography as transition systems and we represented Web services credential disclosure policies as directed graphs. Then, we presented a technique to verify that all possible conversations of the Web services choreography can be implemented by matching credential disclosure policies of the invoker Web services with the access control policy of the Web services being invoked. We have proposed a resource release graph to enable this verification.

References

[1] Ardagna, C., Damiani, E., De Capitani di Vimercati, S., and Samarati,P. A Web Service Architecture for enforcing access control policies. In *Proc. 1st International Workshop on Views on Designing Complex Architectures*, Bertinoro, Italy, September 2004.

[2] Benatallah, B., Casati, F., and Toumani, F. Web service conversation modeling: A cornerstone for e-business automation. IEEE Internet Computing 8, 1, 46–54.

[3] Berardi, D., Calvanese, D., De Giacomo, G, Mecella, M. Composition of Services with Nondeterministic Observable Behavior. In *Proc. Third International Conference on Service Oriented Computing (ICSOC 2005)*, Amsterdam, The Netherlands, December 12-15, 2005.

[4] Busi,N., Gorrieri,R., Guidi,C., Lucchi,R. and Zavattaro, G. Choreography and Orchestration Conformance for System Design. In *Proc. of 8th International Conference on Coordination Models and Languages*, 2006.

[5] Foster, H., Uchitel, S., Magee, J., Kramer, J. WS-Engineer: A Rigorous Approach to Engineering Web Service Compositions and Choreography. In *Proc. of XML 2006 Conference*, Boston, USA, December, 2006.

[6] Kavantzas, N. at al. Web Services Choreography Description Language Version 1.0, W3C Candidate Recommendation, 2005. Online at: http://www.w3.org/TR/ws-cdl-10/.

[7] Kazhamiakin, R., Pistore, M. Choreography conformance analysis:asynchronous communications and information alignment. In *Proc. of Web Services and Formal Methods, Third International Workshop, WS-FM*, 2006.

[8] Mecella, M., Ouzzani, M., Paci, F., Bertino, E. Access Control for Conversation-based Web services. In *Proc. of 15 International World Wide Web Conference (WWW 2006)*, Edinburgh, Scotland, UK, May 2006.

[9] Namli, T., Dogac, A. Using SAML and XACML for Web Service Security & Privacy. A Chapter Proposal, 2007.

[10] Robinson, P., Kerschbaum, F., Schaad, A.: From Business Process Choreography to Authorization Policies. In Proceedings of *20th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, Sophia Antipolis, France, July 31-August 2, 2006.