A System for Interactive Authorization for Business Processes for Web Services[*]

Hristo Koshutanski and Fabio Massacci

Dip. di Informatica e Telecomunicazioni - Univ. di Trento
via Sommarive 14 - 38050 Povo di Trento (ITALY)
{hristo, massacci}@dit.unitn.it

Abstract. Business Processes for Web Services are the new paradigm for virtual organization. In such cross organizational partnerships no business partner may guess a priori what kind of credentials will be sent by clients nor the clients may know a priori the needed credentials for the successful completion of a business process. This requires an interaction between server and clients.

We propose a framework for managing the authorization interactions for business processes and a BPEL4WS based implementation using Collaxa server. Our model is based on interaction between servers and clients and exchange of requests for supplying or declining missing credentials.

Keywords: Web Services, Business Processes For Web Services, Credential-Based Systems, Interactive Authorizations.

1 Introduction

Business Processes (BPs) for Web Services (WS) is the new buzzword for e-commerce integration. BPs allow for a lightweight integration of business partners' services and the establishment of virtual enterprises on the Web. To support this process a number of standards have emerged: SOAP and WSDL for basic functionalities, BPEL4WS and ebXML for complex business processes.

Business Processes are distributed among different partners and all communications are channeled by the invocation of web services by the client. The major difference with traditional access control is that each "task" of the workflow is offered as a web service that can be activated by anyone and thus credentials must be used to enforce access control.

In this paper we discuss our system for reasoning about access control for BPs for WS. The basic intuition is that partners, offering web services in a BP, do not know a priori what credentials clients may need to present nor clients know exactly which services they want, as the BP may take different paths. So, we need an interactive process in which the client starts a business process and the partners evaluate client's current credentials to determine whether they are

[*] This work is partially funded by the IST programme of the EU Commission FET under the IST-2001-37004 WASP project and by the FIRB programme of MIUR under the RBNE0195K5 ASTRO Project and RBAU01P5SS Project.

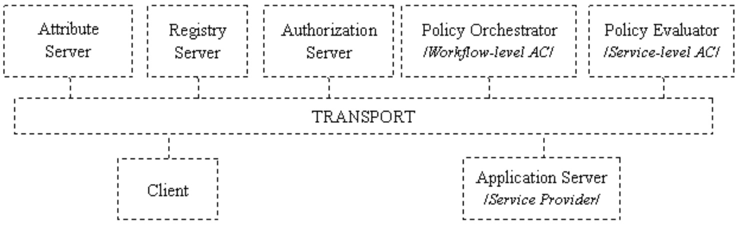

Fig. 1. System Architecture

sufficient or something is missing. Then they get back to the client which may decline some requested credential and a new path must be sought.

We need to find a way for WS partners to find a solution assuming they only know their policies. Further, it does not make sense for a BP to ask all potentially useful credentials (too demanding and privacy intruding for clients) nor such option is practical, considering that WS partners may prefer to ask for some credentials directly to clients rather than making them publicity available.

We have given a semantics to the framework using Datalog, as customary in many approaches to workflow and trust management [1–5]. This allow us to ground the intuitive question: "How does a WS partner determine the credentials needed for granting a request?" into a formally defined procedure.

We have implemented the framework using BPEL4WS server and a front-end to an abduction/deduction engine.

Notice that we are solving a different problem than trust negotiation [5] where both client and server already know what they want and use a sophisticated protocol to disclose each others' credentials to build trust. Trust negotiation could well be applied on top of this framework.

2 System Architecture

In this section we sketch the architecture of the system. We refer to [6] for additional information on the rationale behind the architecture. At the time of writing we have done an initial prototype including the main entities of the system, given below. Figure 1 shows a view of the architecture.

PolicyEvaluator takes endpoint decisions on access control. Each partner is represented by a PolicyEvaluator. It encapsulates the partner's specific authorization policy, and presents it as a service using WSDL.

PolicyOrchestrator is responsible for the workflow level authorization. It decides which partners are involved and according to some orchestration security policies combines the corresponding PolicyEvaluators in a form of a business process executable by the AuthorizationServer.

AuthorizationServer *locates* and *manages* all needed PolicyEvaluators and returns an appropriate result to the ApplicationServer. Also, it is responsible for managing all *interactions* with the Client.

An authorization example of the message flow in our architecture is the following: after an ApplicationServer has been asked for a Web Service by the Client, it requests the AuthorizationServer to confirm whether the Client has enough access rights for that service or not. Then the AuthorizationServer, having client's current credentials and the service request, calls the PolicyOrchestrator for a policy composition process indicating what should be done for taking a decision. Once getting the policy process the AuthorizationServer executes it, communicates with all partners involved and manages their interaction with the Client. When the final decision is taken (grant/deny) the AuthorizationServer informs the ApplicationServer.

3 The Framework

In our framework each partner has a *security policy for access control* $\mathcal{P}_\mathcal{A}$ and a *security policy for disclosure control* $\mathcal{P}_\mathcal{D}$. $\mathcal{P}_\mathcal{A}$ is used for making decision about usage of all web services offered by a partner while $\mathcal{P}_\mathcal{D}$ is used to decide the credentials whose need can be potentially disclosed to the client.

To execute a service of the fragment of BP, under the control the partner, the user will submit a set of *presented credentials* $\mathcal{C}_\mathcal{P}$, a set of *declined credentials* $\mathcal{C}_\mathcal{N}$ and a *service request* r. We assume that $\mathcal{C}_\mathcal{P}$ and $\mathcal{C}_\mathcal{N}$ are disjoint.

The formal model is described at length in [7], in Figure 2 we show the summary of the algorithm.

We use the symbol $P \models L$, where P is a policy and L is either a credential or a request to specify that L is a logical consequence of a policy P. P is *consistent* ($P \not\models \bot$) if there is a model for P. Abduction solution (step 3b of algorithm in Fig. 2) over a policy P, a set of predicates H with defined p.o. over subsets of H and a ground literal L is a set of ground atoms E such that: (i) $E \subseteq H$; (ii) $P \cup E \models L$; (iii) $P \cup E \not\models \bot$; (iv) any set $E' \prec E$ does not satisfy all conditions above; Traditional p.o.s are subset containment or set cardinality.

The use of declined credentials is essential to avoid loops in the process and to guarantee the success of interaction in presence of disjunctive information. For example suppose we have alternatives in the partner's policy (e.g., "present either a VISA or a Mastercard or an American Express card"). An arbitrary alternative can be selected by the abduction algorithm and on the next interaction step (if the client has declined the credential) the abduction algorithm is informed that the previous solution was not accepted. The process can continue until all credentials have been declined (and access is denied) or a solution is found (and access is granted). Additional details on the formal model can be found in [7].

4 Implementation

For our implementation, Collaxa[1] is used as a main BPEL manager (on the AuthorizationServer side) for executing and managing all policy composition pro-

[1] Collaxa BPEL Server (v2.0 rc3) – www.collaxa.com

```
1. extract from the client's input the set of presented credentials $\mathcal{C}_{\mathcal{P}}$ and
   the set of declined credentials $\mathcal{C}_{\mathcal{N}}$
2. verify that the request is a logical consequence of the credentials, namely
   $\mathcal{P}_{\mathcal{A}} \cup \mathcal{C}_{\mathcal{P}} \models r$
3. if the check succeeds then access is granted, otherwise
   (a) compute the set of disclosable and not declined credentials as
       $\mathcal{C}_{\mathcal{D}} = \{c \mid c \text{ credential that } \mathcal{P}_{\mathcal{D}} \cup \mathcal{C}_{\mathcal{P}} \models c\} \setminus \mathcal{C}_{\mathcal{N}}$
   (b) use abduction to find a minimal set of missing credentials $\mathcal{C}_{\mathcal{M}} \subseteq \mathcal{C}_{\mathcal{D}}$
       such that both $\mathcal{P}_{\mathcal{A}} \cup \mathcal{C}_{\mathcal{P}} \cup \mathcal{C}_{\mathcal{M}} \models r$ and $\mathcal{P}_{\mathcal{A}} \cup \mathcal{C}_{\mathcal{P}} \cup \mathcal{C}_{\mathcal{M}} \not\models \bot$
   (c) if no such set exists then $\bot$ is sent back to the user, otherwise
   (d) communicate $\mathcal{C}_{\mathcal{M}}$ back to the client and iterate the process.
```

Fig. 2. Interactive Access Control for Stateless WS

cesses returned by the PolicyOrchestrator and for the implementation of the AuthorizationServer itself. Some of Collaxa's main characteristics:
– it supports many WS standards as BPEL4WS, WSDL, SOAP, etc;
– it interoperates with platforms as BEA's WebLogic and Microsoft .NET;
– it is easy to integrate Java modules (classes) within a BPEL process;
– deploying a process on Collaxa is actually compiling it down to Java code that is internally executed by the JVM when invoked.

The AuthorizationServer itself is a BPEL process deployed under Collaxa that internally deploys the policy process returned by the PolicyOrchestrator as an internal web service and internally executes it. The advantage is that if the AuthorizationServer is requested to get an access decision for a service that has already been asked for it and there is no change in the workflow policy then the AuthorizationServer *does not* deploy the service's policy process again but just (internally) executes it. In that way we speed up the access decision time by *JIT compilation of authorization processes.*

PolicyOrchestrator in the current prototype is just a mapping between a service resource and its workflow policy process. We assume that the process is already created by some GUI (e.g., could be used any BPEL visual tool generator) and is available to the orchestrator.

PolicyEvaluator is a Java module that acts as a wrapper for the DLV system[2] (a disjunctive datalog system with negations and constraints) and implements our interactive algorithm for stateless WS (Fig. 2). For deductive computations we use the disjunctive datalog front-end (the default one) while for abductive computations, the diagnosis front-end.

[2] DLV System (r. 2003-05-16) – www.dlvsystem.com

5 Future and ongoing work

There are a number of issues that are currently the subject of research to improve the usability of our system. Following are the key points in our future and ongoing work.

The current system processes credentials at an high level: defines what can be inferred and what is missing from a partner's access policy and a user's set of credentials. There is the need of a suitable platform for the actual distributed management of credentials at lower levels (namely actual cryptographic verification of credentials). We decided to use PERMIS infrastructure [8] because it incorporates and deals entirely with X.509 Identity and Attribute Certificates. It allows for creating, allocating, storing and validating such certificates. Since PERMIS conforms to well-defined standards we can easily interoperate with the other entities (partners) in a BP.

Next step in the framework is to use algorithms for credentials' chain discovery as in [9]. Then, once a client collects all credentials and supplies them back to the service provider, it can be used, as a preprocessing step (before running the access control procedure), for tracking all credentials provided by him.

Since we are at an initial stage of our prototype, we have only run limited experiments. Substantial large scale experiments are yet to be worked out to determine the running performance of the algorithm. Cassandra, a role based policy language, and its policies for the UK's EHR could be a good candidate for a benchmark [10].

References

1. Li, N., Mitchell, J.C., Winsborough, W.H.: Design of a role-based trust-management framework. In: Proc. of IEEE SS&P. (2002)
2. Kang, M.H., Park, J.S., Froscher, J.N.: Access control mechanisms for inter-organizational workflow. In: Proc. of ACM SACMAT. (2001) 66–74
3. Bertino, E., et al.: The specification and enforcement of authorization constraints in workflow management systems. ACM TISSEC **2** (1999) 65–104
4. Atluri, V., Chun, S.A., Mazzoleni, P.: A Chinese wall security model for decentralized workflow systems. In: Proc. of ACM CCS. (2001) 48–57
5. Yu, T., Winslett, M., Seamons, K.E.: Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. ACM TISSEC **6** (2003) 1–42
6. Koshutanski, H., Massacci, F.: An access control framework for business processes for Web services. In: Proc. of ACM Workshop on XML Security. (2003)
7. Koshutanski, H., Massacci, F.: Interactive access control for Web Services. In: the 19th IFIP Information Security Conference (SEC). (2004) (to appear).
8. Chadwick, D.W., Otenko, A.: The PERMIS X.509 role-based privilege management infrastructure. In: the 7th ACM SACMAT. (2002) 135–140
9. Li, N., Winsborough, W.H., Mitchell, J.C.: Distributed credential chain discovery in trust management. Journal of Computer Security **11** (2003) 35–86
10. Becker, M.Y., Sewell, P.: Cassandra: Flexible trust management, applied to electronic health records. In: Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW). (2004) (to appear).