An Access Control System for Business Processes for Web Services

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

June 2003 – Submitted to NORDSEC

Abstract

Web Services and Business Processes for Web Services are the new paradigms for the lightweight integration of business from different enterprises.

Security and access control policies for Web Services protocols and distributed systems are well studied and almost standardized, but there is not yet a comprehensive proposal for an access control architecture for business processes. The major difference is that business processes describe complex services that cross organizational boundaries and are provided by entities that sees each other as just partners and nothing else.

This calls for a number of differences with traditional aspects of access control architectures such as: credential vs. classical user-based access control; interactive and partner-based vs. one-server-gathers-all requests of credentials from clients; controlled disclosure of information vs. all-or-nothing access control decisions; abducing missing credentials for fulfilling requests vs. deducing entailment of valid requests from credentials in formal models.

Looking at the access control field we find good approximation of most components but not their synthesis into one access control architecture for business processes for web services, which is the contribution of this paper.

1 Introduction

Middleware has been the enterprise integration buzzword at the end of the past millennium. Nowadays a new paradigm is starting to take hold: Web Services (WS for short). Setting hype aside, the major difference between middleware solutions (CORBA, COM+, EJB, etc.) and WS is the idea of lightweight integration of business processes from different enterprises.

Basic WS are well studied and standardized, for what concerns access control and security. There are also many approaches [16, 3, 4] for controlling access to services in distributed systems, and an advanced standardization process (see for instance the OASIS XACML [8] proposal). With the notable exception of provisional access control [10] and trust negotiation [17], access control models rest on the idea that the server picks the evidence you sent on who you are (credentials), and what you want (request), checks its evidence on what you deserve (policies) and makes a decision.

Moving up in the WS hierarchy from single services to *orchestration and choreography of WS and business processes* the picture changes. Business processes describe complex services that cross organizational boundaries and are provided by partners. See Appendix B for a primer on WS and business processes.

The paradigmatic example in the WS standards is a travel agent WS that must orchestrate a combination of plane and train tickets, car rental, hotel booking and insurance, each service offered by different partner which may or may not be involved according to the actual unrolling of the workflow.

For example consider the problem of going to a nice "Shakespearian Tour" in Italy: you might decide to go to the city of Shylock, and from there rent a car and travel to Romeo and Juliet's last resort, to jump then on a train and visit the Senate's seat where Pompeous spoke after Caesar's death. However, you might as well decide to travel instead to Germany first and then the train to Verona from there. In the first case you might need to use a car rental company. The second path may require to contact a German train company for the schedule, which is not needed if you land directly in Italy.

Let us now consider the problem of "lightweight" credentials such as the German train discount card or the car rental gold member card. Should the user provide them anyway at the beginning? Obviously not. Should the server orchestrating the process require each partner to publish its policy on discounts? Obviously not. Such problems are not simply problems of practicality, but have major security implications:

1. Credential vs. identity based access control – A WS is something you publish on the Web for everybody to use it, so the system has to be close to trust management systems [4];

2. Orchestrating vs. combining – *partners* have different security policies and *are just partners* and not part of the same enterprise. They may not wish to disclose their policies to the server orchestrating the request. So, we cannot simply combine the policies, we need to orchestrate the request grant/deny/process of many different policies/partners.

3. Interactive vs. one-off access control – if partners have different policies they might as well require different credentials to a client. Privacy considerations make gathering all potentially needed credentials from clients difficult. Furthermore, this may simply be impossible. An airline may want to ask confidential information directly to its frequent fliers (e.g., confirmation of religious preferences for the food) and not to the Web travel agent orchestrator of the process. This calls for an interactive process in which the client may be asked on the fly for additional credentials and may grant or deny such requests[1].

4. Abducing vs. deducing credentials – in most classical formal models we deduce that a request is valid because it is entailed by the combination of the policy and the set of available credentials. Here, a partner must be able to infer the causes of some failed request to ask the missing credentials to the client. The corresponding logical process is no longer deduction but it is abduction. So we

[1]Note that the workflow may even take completely different paths based on the results of interaction. For example a rent-a-car operator may require a signed credit card number plus a physical address. The client may deny such requirement and thus another operator may be chosen that only asks for a credit card number.

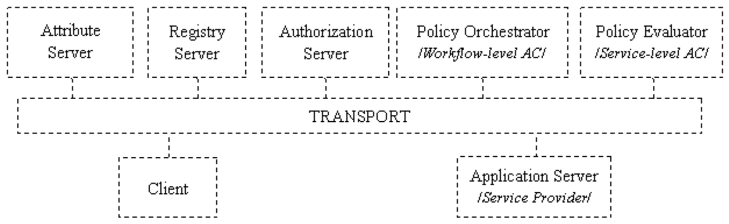

Attribute Server
Registry Server
Authorization Server
Policy Orchestrator /Workflow-level AC/
Policy Evaluator /Service-level AC/
TRANSPORT
Client
Application Server /Service Provider/

Figure 1: Cross-section view of the architecture

must have co-existence of deduction (for deciding access and release of information) and abduction (for explaining failed accesses).

5. Data vs. source level communication – the choice of format for messages is always rather complicated, as it calls for the implementation of software that is able to interpret its meaning. In a Business Process scenario we no longer need messages, but just "mobile" processes. A client will receive a business process so that he can simply execute the source to obtain and send the missing credential. An authorization server can download a business process from a policy orchestrator and obtain the desired authorization.

Looking at the access control field we find a good approximation of most components: we have proposals for combining policies at the logical level [11, 15] and at the architectural level [8]. We have proposals for calculi for controlling release of information [5], and procedures for trust negotiations and communication of credentials [17], architecture for distributed access control [8, 3, 16].

What is missing is a way to synthesize *all* these aspects into one access control architecture for business processes of WS, which is the contribution of this paper.

Next section presents our architecture and discusses how the entire message passing scheme can be implemented as "mobile" processes. Section 4 explains how we can use logical deduction and logical abduction to build a firm foundation for the interactive process of inferring disclosable credentials from access control policies and from release policies. Next we discuss how everything can be implemented using Business Process themselves. A brief discussion of related works concludes the paper.

2 Architecture

Combining the traditional proposals for distributed access control and the essential components used for Web services we propose here a security architecture for orchestrating authorization of Web Services Processes. Figure 1 shows a cross-section view of the architecture, whereas Figure 2 shows a horizontal view of it. A brief description of the servers shown in the figure is given below.

AttributeServer is responsible for providing group/role membership information as in [16], for instance in the form of membership and non-membership certificates.

RegistryServer is responsible for maintaining relations between services and service providers implementing a particular service. When a Client requests the RegistryServer for a specific service, the latter responds with a list of ApplicationServers implementing the requested service.

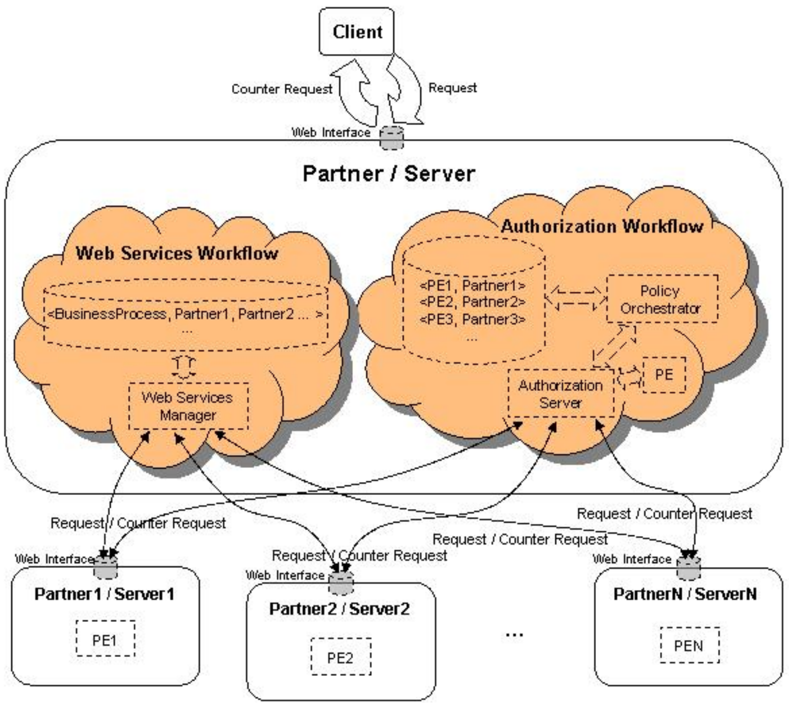


Figure 2: Horizontal view of the architecture

AuthorizationServer decouples the authorization logic from the application logic. It is responsible for *locating*, *executing*, and *managing* all needed PolicyEvaluators, and returning an appropriate result to the ApplicationServer. Also it is responsible for managing all the *interactions* with the Client.

PolicyEvaluator terminology borrowed from Beznosov et al [3], is an entity responsible for achieving endpoint decisions on access control (see Figure 1). All partners involved in a business process are likely to be as different entities, each of them represented by a PolicyEvaluator.

PolicyOrchestrator from the authorization point of view is an entity responsible for the workflow level access and release control. It decides which are the partners that are involved in the requested service (Web service workflow) and on the base of some orchestration security policies to combine the corresponding PolicyEvaluators in a form of a Web process (*Policy Composition Process*) that is suitable for execution by the AuthorizationServer.

To secure the entire architecture we must make some assumptions on the security properties of the lower levels. Obviously we assume authentication, confidentiality, and message integrity at the transport and message levels. So, we assume that we have already in place the proposed standards.

At transport level we assume the adoption of the WS-Security specification[2] that describes enhancements to SOAP messaging to provide message integrity, confidentiality, and authentication. For the message level one can use the W3C and IETF specification for XML-Signature[3] and W3C XML-Encryption[4], or the recently release specifications by IBM and Microsoft for WS secure conversations[5].

[2]WS-Security – http://www-106.ibm.com/developerworks/webservices/library/ws-secure/
[3]XML-Signature – http://www.w3.org/TR/xmldsig-core/
[4]XML-Encryption – http://www.w3.org/TR/xmlenc-core/
[5]WS-SecureConversation – http://www.ibm.com/developerworks/library/ws-secon/

Assuming security at lower level, the second key component is the languages and format of communications. We propose here a major innovation: the typical exchange of messages in an access control system is at "data" level (credentials, policies, requests, objects, etc.) that are interpreted by the recipients. This choice makes the actual implementation of proposed access control infrastructure difficult and often not easily portable. Here we propose to exchange messages at "source code" level and in particular at the level of business process description. It means that instead of sending just messages that have to be interpreted by entities, we truly have mobile processes passing from one entity to another indicating themselves what the recipient has to do.

The mobility of authorization processes has a number of advantages. First of all a server simply needs an off-the-shelf interpreter for business processes for a quick implementation. Second we have more flexibility for describing the process leading to an access control decision. Some PolicyEvaluators may decide to disclose it XACML policies and therefore send a mobile processes, which just describe the evaluation of the policies along some XACML rules. Other PolicyEvaluators may instead decide to offer an external interface, so that they just specify a container for requests and an output container for its decision. All intermediate choices are possible so that one can accommodate also provisional access control or the interactive version that we advocate here.

Leading this approach at an extreme the AuthorizationServer can simply receive a business process from the orchestrator and execute it. The process may still be computationally intensive as an AuthorizationServer may have to process thousands or millions of authorization workflows, but it could be logically very simple thus reducing the TCB to the simple execution of certified processes from certified sources[6].

The role of the PolicyEvaluator is to encapsulate the connected with it partner's specific access control model, authorization policy, and requirements with their internal representation, interpretation, and mechanisms for computing an access decision and presenting it as a service using standardized Web service interface (e.g., WSDL).

The entity burdened with constructing the authorization workflow (Figure 2) is the PolicyOrchestrator. The PolicyOrchestrator functionality can be considered as having two main tasks: first one, called *Policy Composition Service*, is to select which are the partners involved in the requested process and to combine the corresponding PolicyEvaluators in a policy composition process, and return it back to the AuthorizationServer. After the AuthorizationServer having finished the execution of the policy composition process it asks[7] the PolicyOrchestrator for applying the workflow level release policies over the results from the execution – the second main task. The process of applying release control polices, called *Release Policy Service*, captures how the final authorization decision should be released to the Client.

For a detailed example of how actors in our framework communicate each other see Appendix A.

3 Interactive Communications as "Mobile" Processes

We have decided to use the term *mobile process* because it well expresses the idea of using mobile code together with the functionality of Web processes. The main advantages of using mobile processes in our authorization framework are *flexibility* and *simplicity* of entities. Flexibility because of recipient of

[6]Recall that we assume that authentication, integrity, and confidentiality are assured at message and transport level.

[7]This is the case if it is specified in the policy composition process, i.e. depends on the security policies being applied in constructing the policy composition process.

mobile process is not limited to the functions and computational algorithms that the recipient's logic predefines. Migrations of actors in the system from one server to another is easier with mobile processes and the system as a whole is more flexible. Entities in the framework becomes simpler, having little functionality pre-engineered into them, as we will see in section 5.

The next important step in advocating mobile processes is to specify a language that is needed for coding them. We have identified it as a *language for communicating interactive requests back to a* Client. This is even in the case when a Client is an AuthorizationServer waiting for a response either from a PolicyOrchestrator or from a PolicyEvaluator. This language can be designed with a black box view of the PolicyEvaluator, but must be easily interpretable from the Client side. Thus we propose to use BPEL4WS itself as a language in which requests are coded. The PolicyEvaluator/PolicyOrchestrator must represent its request as a WS business process that can then be interpreted and executed by the Client. If the PolicyEvaluator wants part of the request to be only visible to the Client it can use the available XML-crypto features to protect the relevant part.

Loosely speaking we may say that the Client starts by executing a simple `<invoke>R</invoke>` and obtain in return either its result or a more complicated process to execute. For example a BPEL4WS interactive request may specify a `<input container>` where to put a digitally signed copy of the travel contract sealed with the public key of the rent-a-car company (a process that can be specified as a `<sequence>` of events).

The idea is intuitive and appealing but there is an essential detail that must be taken care of. Notably, the AuthorizationServer will receive a number of interactive requests while controlling its workflow and the combination of these requests and the service workflow specification is essential. The simplest solution is to ignore such interaction: all interactive requests are compiled into a `<flow>` and the result is sent back to the Client. Such solution is hardly satisfactory from the point of view of the Client: we often want to know "why" some additional information is needed. See the example of Figure 8: at some stage somebody may ask for a digitally signed declaration about our address. We may consider this request fair enough from the shipping agent, but not from the credit checking bureau. So, each BPEL4WS interactive request must be supplemented with a special tag [root/context]:

- root requests will be compiled with a `<flow>` construct and returned together with the overall result of the computation for contextual requests;
- contextual requests the PolicyOrchestrator will make a copy of the WS process (*not* the authorization process) and replace each step S for which an additional request I has been called with the request and a context indicating the WS (partner and all) that required the additional credential. The PolicyOrchestrator will then prune the WS process removing all nodes that were not on a path from the root to the newly modified nodes and sends the result to the Client.

The last step is necessary to protect the overall workflow from unnecessary disclosure.

This combination is sufficiently adequate for most uses, but still it offers the PolicyOrchestrator just the choice of compiling individual requests rather than combining them. Here we have identified an important point in the PolicyOrchestrator where we need to introduce a new language - a *language for combination of policies and interactive requests at workflow level.* So far we have not found a proposal that is entirely satisfactory, part because there are not enough case studies of WS Business Processes to guide the selection of policies at workflow level.

The proposal by Bertino et al. [2], is fairly expressive but only focuses on implementing snapshot constraints on a workflow level (i.e. safety properties). So it is not possible to express properties such as "if Y is repeatedly true then eventually X should happen".

The usage of algebraic constructs based on dynamic logic proposed by Wijesekera and Jajodia [15] seems more promising. Indeed `<invoke>` operation would be mapped into single action, `<sequence>` into sequential compounder, `<switch>` into non deterministic choice (each case represented by a test) and `<flow>` by intersection. This does not mean that we would use dynamic logic for actual implementation[8], but rather that the logical language may offer a formal foundation to policy written in BPEL4WS.

4 The Abduction of Missing Credentials

For the deployment of the architecture, the PolicyEvaluator must be able to determine the set of additional credential that are necessary to obtain a service in case of failure. This problem may of course be shifted on the implementors of PolicyEvaluators, as the architecture only needs that the outcome of this derivation is mapped into some BPEL4WS process that is then sent to the client.

However, there is no algorithm in either the formal or the practical models of access control and trust negotiations to derive such credentials from the access control policy. The works on trust negotiations [13, 17] focus on communication and infrastructure and assume that requests and counter requests can be somehow calculated from the access policy. The formal models on credential-based access control and policy combination [2, 11, 15] don't treat the problem of inferring missing credentials from failed requests, as they are within the frame of mind of inferring successful requests from present credentials. Also standardization efforts like the XACML proposals [8] gives rules for deriving what is right (evaluating policies) and not rule for understanding what is wrong.

Here, we present an approach based on logic that allows for a clean solution of these problems. For sake of simplicity (and popularity), assume that the policy is expressed using Datalog rules or logic programs with the stable model semantics (if we need negation to implement some constraints like separation of duties). What we need is a logical implementation of the following process:

1. the PolicyEvaluator receives the credentials and evaluates the request against the policy augmented with the credentials, i.e. whether the request is a logical consequence of the policy and the credentials;

2. if the request is granted nothing needs to be done;

3. if the request fails we evaluate the given credential against a release policy of the PolicyEvaluator to infer which are the credentials whose need can be disclosed on the basis of the credentials already received;

4. abduce the actually needed credentials by re-evaluating the request against the policy and considering the potentially disclosable credentials determined at the previous step; only the needed credential are communicated to the client.

[8]This is less critical than prejudice may suggest. The ML implementation of Peter Patel-Schneider at Bell-Labs can actually crack significant dynamic logic theorems in milliseconds.

In a nutshell, what we need for the implementation of PolicyEvaluator is to implement two main inference capabilities: *deduction* and *abduction* [14]. We need to use deduction to infer whether a request can be granted on the basis of the present credentials as in [5, 2, 11], we use abduction to explain which minimum set of credentials would be necessary to grant a failed request. Obviously it is not necessary to use logic, what we claim is that the underlying logical constructs that we need for our access decisions are these two conceptually different operations.

Due to lack of space, here we just give the basic hint of the formalization.

Definition 1 (Access Control) *Let P be a datalog program (or stratified logic program) representing an access control policy, let r be an atom representing a request, let C be a set of atoms representing a set of given credentials, the* request is granted *if and only if $P \cup C \models r$.*

Definition 2 (Release Control) *Let P be a datalog program (or stratified logic program) representing a release control policy, let d be an atom representing a credential, let C be a set of atoms representing a set of given credentials, the* credential d is disclosable *if and only if $P \cup C \models d$.*

Definition 3 (Access Control Explanation) *Let P be a datalog program (or stratified logic program) representing an access control policy, let r be an atom representing a request, let C be a set of atoms representing a set of given credentials, let $D_P \supseteq C$ be a set of atoms representing disclosable credentials, an* explanation of missing credentials $C_M \subseteq C_P$ *such that*

1. *$P \cup C \not\models r$*
2. *$P \cup C \cup C_M \models r$*
3. *$P \cup C \cup C_M$ is consistent*

The first conditions says that the missing credentials are indeed needed. The second condition says that they are sufficient and the last condition says that they are actually meaningful. In presence of positive Datalog program such as for Bonatti and Samarati's logic [5] and Li's Delegation Logic 1 [11], the consistency condition is satisfied by default. In presence of constraints on the execution or negation as failure, as in Bertino et al. Datalog programs for workflow policies [2] — which can be easily augmented with credentials — the consistency condition is essential to guarantee that the abduced set of atoms makes sense. Indeed, constraints could make $P \cup C \cup C_M$ inconsistent and therefore it would not make much sense to say that the request r should be granted from a system.

In Figure 3 is shown a logic program showing a university online library access and release rules. The notations for declarations, credentials, and services are borrowed from Bonatti and Samarati [5]. Here `decl` means that it is a statement (e.g., identity, address) declared by the client, while `cred` is a statement declared and signed by a key corresponding to some trusted authority. Consider rule 4 that says "to have access to service `reading` the client should have access to library (presenting Id and some library card) and a loan library card". Rule 10 says "to reveal the need for a loan library credential there should be a declaration of the library's Id and some library credential".

If the PolicyEvaluator is given the declaration decl$(id1568)$ and the credential cred$(card(user, john, id1568), bibK)$, together with the request for reading the journal articles on-line. The query serv$(reading)$ does not follow from the policy and the given declarations and credentials. So, we apply the release policy and infer

Access Policy:

$$\begin{aligned}
\text{serv}(\mathit{query}()) &\leftarrow \text{decl}(\mathit{Id}), \text{cred}(\mathit{card}(\mathit{Type}, \mathit{Name}, \mathit{Id}), \mathit{biblioK}) & (1)\\
\text{serv}(\mathit{query}(\mathit{citations})) &\leftarrow \text{serv}(\mathit{access}), \text{cred}(\mathit{member}(\mathit{Name}, \mathit{Dept}), K_D), \mathit{assoc}(\mathit{Dept}, K_D) & (2)\\
\text{serv}(\mathit{booking}) &\leftarrow \text{decl}(\mathit{Name}, \mathit{Dept}), \text{cred}(\mathit{card}(\mathit{loan}, \mathit{Name}, \mathit{Id}), \mathit{biblioK}) & (3)\\
\text{serv}(\mathit{reading}) &\leftarrow \text{serv}(\mathit{access}), \text{cred}(\mathit{card}(\mathit{loan}, \mathit{Name}, \mathit{Id}), \mathit{biblioK}) & (4)\\
\text{serv}(\mathit{reading}) &\leftarrow \text{cred}(\mathit{academic}(\mathit{Name}, \mathit{UnivId}), K_U), \mathit{assoc}(\mathit{university}, K_U) & (5)\\
\text{serv}(\mathit{reading}) &\leftarrow \text{serv}(\mathit{query}(\mathit{citations})), \text{cred}(\mathit{researcher}(\mathit{Name}, \mathit{Dept}), K_D), \mathit{assoc}(\mathit{Dept}, K_D) & (6)
\end{aligned}$$

Release Policy:

$$\begin{aligned}
\text{decl}(\mathit{Name}, \mathit{Dept}) &\leftarrow \text{decl}(\mathit{Id}) & (7)\\
\text{cred}(\mathit{researcher}(\mathit{Name}, \mathit{Dept}), K_D) &\leftarrow \text{decl}(\mathit{Name}, \mathit{Dept}), \text{cred}(\mathit{card}(\mathit{Type}, \mathit{Name}, \mathit{Id}), \mathit{bibK}) & (8)\\
\text{cred}(\mathit{member}(\mathit{Name}, \mathit{Dept}), K_D) &\leftarrow \text{decl}(\mathit{Name}, \mathit{Dept}) & (9)\\
\text{cred}(\mathit{card}(\mathit{loan}, \mathit{Name}, \mathit{Id}), \mathit{bibK}) &\leftarrow \text{decl}(\mathit{Id}), \text{cred}(\mathit{card}(\mathit{Type}, \mathit{Name}, \mathit{Id}), \mathit{bibK}) & (10)\\
\text{cred}(\mathit{academic}(\mathit{Name}, \mathit{UnivId}), K_U) &\leftarrow \text{decl}(\mathit{UnivId}), \text{decl}(\mathit{Name}, \mathit{Dept}) & (11)
\end{aligned}$$

Figure 3: University Library WS Access and Release Policies

that the following credentials are disclosable:

$$\text{decl}(\mathit{john}, \mathit{cs}), \text{decl}(\mathit{id}1568), \text{cred}(\mathit{researcher}(\mathit{id}1568, \mathit{cs}), \mathit{csK}), \text{cred}(\mathit{card}(\mathit{user}, \mathit{john}, \mathit{id}1568), \mathit{bibK}),$$
$$\text{cred}(\mathit{member}(\mathit{john}, \mathit{cs}), \mathit{csK}), \text{cred}(\mathit{card}(\mathit{loan}, \mathit{john}, \mathit{id}1568), \mathit{bibK}).$$

The abduction algorithm derive two possible answers for the credentials:

$$\begin{aligned}
C_{M1} &= \{\text{decl}(\mathit{john}, \mathit{cs}), \text{cred}(\mathit{member}(\mathit{john}, \mathit{cs}), \mathit{csK})\}\\
C_{M2} &= \{\text{cred}(\mathit{card}(\mathit{loan}, \mathit{john}, \mathit{id}1568), \mathit{bibK})\}
\end{aligned}$$

Both sets are minimal with respect to the subset inclusion ordering and only C_{M2} is minimal with respect to a set cardinality ordering. In case the first set is chosen the PolicyEvaluator will compile a `<flow>` node for sending the requests back to the client.

5 Component Algorithms as Business Processes

This section shows how we can describe entities in our architecture and how they can communicate each other using BPEL4WS specification (see Appendix B).

The Client and ApplicationServer processes are shown in Figure 4. In the figure on the left, after the Client has requested the ApplicationServer for getting a service R, presenting its credentials, there are two cases: Additional Request - in this case is returned a counter request (a process), indicating what should be done by the Client. After that locally is invoked a service *DoAddRequestService* for executing the required process. Because of the while loop again is requested the service R with the result of the process; ResultOfOperation - in this case is returned the result of the requested service R and the Client's process finishes. The ApplicationServer, after the Client's request for accessing the service R, asks the RegistryServer (step 1 in Figure 4 on the right) for locating its *AuthorizationService*. After that the

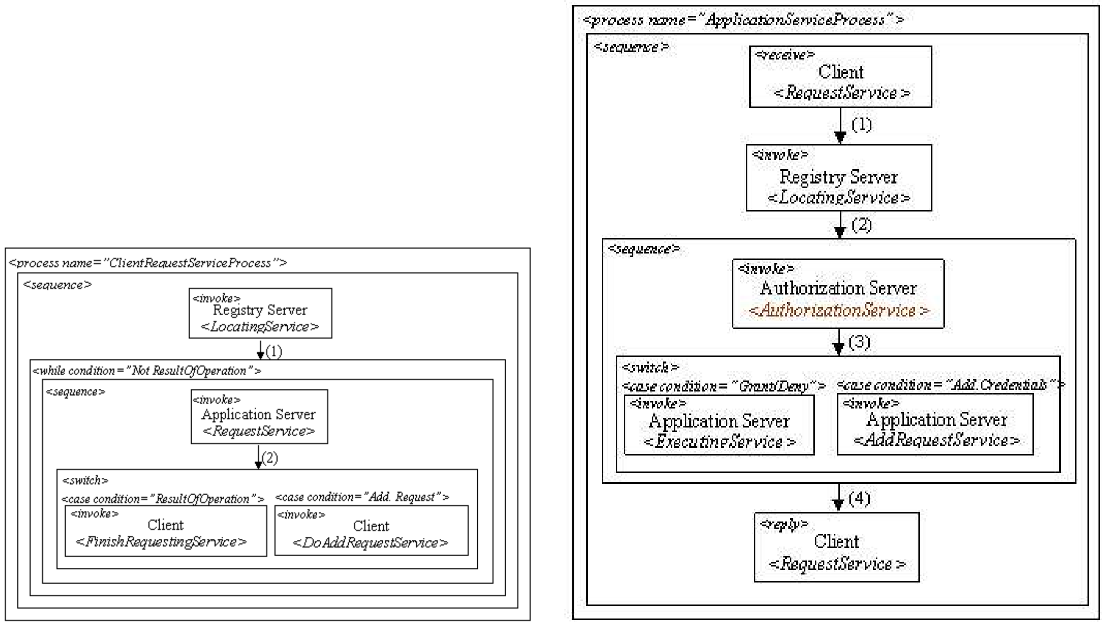


Figure 4: Client and Application Server Process Diagrams

AuthorizationService is invoked along with Client's credentials and the requested service R for taking the authorization decision (step 2 in Figure 4). Then we can switch between explicit Grant/Deny response returned from the AuthorizationServer in the case of which is executed or not the requested service R and the results are returned back to the Client (step 4 in Figure 4), or in the case of additional credentials is executed the *AddRequestService*, which either executes some counter-requirements that have to be presented to the Client or redirects the entire request to the Client (step 4 in Figure 4).

The AuthorizationServer process, shown in Figure 5, is the following: after the *AuthorizationService* has been invoked by the ApplicationServer the *PolicyCompositionService* located in the PolicyOrchestrator is invoked. The result of the service invocation (step 1 in Figure 5) is a policy composition process (e.g., BPEL4WS) indicating what should be done by the AuthorizationServer in order to be taken the final authorization decision. After obtaining the process (step 2 in Figure 5), the AuthorizationServer starts executing it, requesting all needed PolicyEvaluators with respect to that process, i.e. some of them in parallel, others in a sequence etc. Here the policy composition process consists of a sequence indicating that first the AuthorizationServer has to execute all PolicyEvaluators relevant to the requested service R orchestrated in a specific way (where the most intuitive structure is a <`flow`> one indicating execution in parallel, as shown in Figure 5), and after that executing the *ReleasePolicyService* responsible for taking the final access decision. After finishing the policy composition process, the AuthorizationServer returns the final access decision to the ApplicationServer (step 4 in Figure 5).

6 Conclusions and Related Work

As we have already discussed, a number of access control models have been proposed for workflows [2], role based access control on the web [7], entire XML documents [1, 6], tasks [9], and DRM [12], possibly coupled by sophisticated policy combination algorithms . However, they have mostly remained within the classical framework where servers know their clients pretty well: they might not know their names

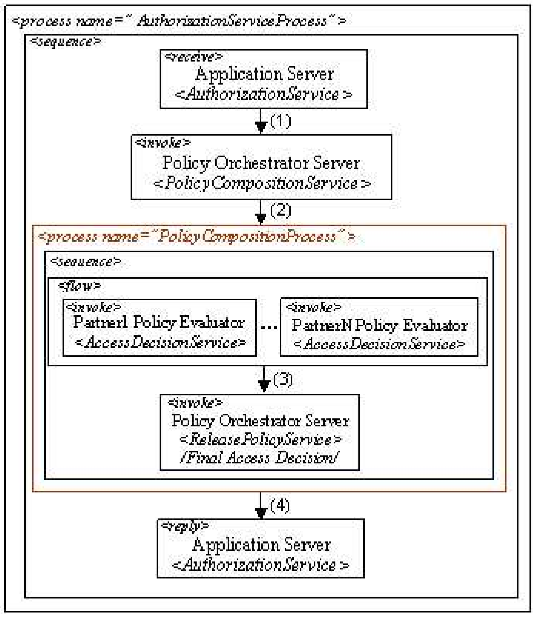


Figure 5: Authorization Server Process Diagram

but they know everything about what, when, and how can be used by these clients.

In most proposals, the possibility that servers may get back to the calling Clients with some counter requests is not considered. This even in the case where the Client is actually an AuthorizationServer querying different PolicyEvaluator servers.

In one of the earliest work on distributed access control by Woo and Lam [16] the ApplicationServer offloads its authorization policy to an AuthorizationServer. After evaluating the policy the Authorization-Server hands out authorization certificate to the Client, which the Client has to present along with its request.

An architecture close to ours has been proposed by Beznosov et al. [3]. Authorizations are managed by an Authorization Service, and its Access Decision Object (ADO). The ADO obtains references to all PolicyEvaluators related to the Client's request, asks a decision combinator for combining decisions according to a combination policy, and returns the decision back to the Client.

In this paper we have proposed a solution to address the challenges of WS processes: a possible architecture for the authorization of business processes for Web services. We have identified an interactive access control model as a way for protecting security interests wrt disclosure of information and access control of both servers and clients. Logical abduction is the solid semantical foundation upon which interaction can be build.

In the model a Client interacts (contracts) with the servent in order to finalize the necessary set of credentials needed to satisfy all partners' requirements related to the process. We propose to use "mobile" processes as messages exchanged in the architecture, and specified how entities in the architecture can be implemented using WS processes themselves.

References

[1] Bertino, E., Castano, S., and Ferrari, E. On specifying security policies for Web documents

with an XML-based language. In *Proceedings of the Sixth ACM SACMAT* (2001), pp. 57–65.

[2] Bertino, E., Ferrari, E., and Atluri, V. The specification and enforcement of authorization constraints in workflow management systems. *ACM TISSEC 2*, 1 (1999), 65–104.

[3] Beznosov K., et al. A resource access decision service for CORBA-based distributed systems. In *Proceedings of 15th IEEE ACSAC* (1999), pp. 310–319.

[4] Blaze, M., Feigenbaum, J., Ioannidis, J., and Keromytis, A. D. The role of trust management in distributed systems security. In *Secure Internet programming: security issues for mobile and distributed objects* (1999), Springer-Verlag, pp. 185–210.

[5] Bonatti, P., and Samarati, P. A unified framework for regulating access and information release on the Web. *Journal of Computer Security*. (to appear).

[6] Damiani, E., di Vimercati, S. D. C., Paraboschi, S., and Samarati, P. A fine-grained access control system for XML documents. *ACM TISSEC 5*, 2 (2002), 169–202.

[7] Giuri, L. Role-based access control on the Web. *ACM TISSEC 4*, 1 (2001), 37–71.

[8] Godik, S., and Moses, T. *eXtensible Access Control Markup Language (XACML)*. OASIS, February 2003. www.oasis-open.org/committees/xacml/.

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

[10] Kudo, M., and Hada, S. XML document security based on provisional authorization. In *Proceedings of the 7th ACM CCS* (2000), pp. 87–96.

[11] Li, N., Grosof, B. N., and Feigenbaum, J. Delegation logic: A logic-based approach to distributed authorization. *ACM TISSEC 6*, 1 (2003), 128–171.

[12] Park, J., and Sandhu, R. Towards usage control models: beyond traditional access control. In *Seventh ACM SACMAT* (2002), pp. 57–64.

[13] Roscheisen, M., and Winograd, T. A communication agreement framework for access/action control. In *Proceedings of the IEEE Symposium on Security and Privacy* (1996), pp. 154–163.

[14] Shanahan, M. Prediction is deduction but explanation is abduction. In *Proceedings IJCAI '89* (1989), Morgan Kaufmann, pp. 1055–1060.

[15] Wijesekera, D., and Jajodia, S. Policy algebras for access control the predicate case. In *Proceedings of the 9th ACM CCS* (2002), pp. 171–180.

[16] Woo, T. Y. C., and Lam, S. Designing a distributed authorization service. In *Proceedings of INFOCOM* (1998), vol. 2, IEEE Press, pp. 419–429.

[17] Yu, T., Winslett, M., and Seamons, K. E. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM TISSEC 6*, 1 (2003), 1–42.

APPENDIX

A An Authorization and Data Flow Example

This section shows an authorization example of the message flow in our architecture, shown in Figure 6. The following example describes how the architectural components compute an authorization decision:

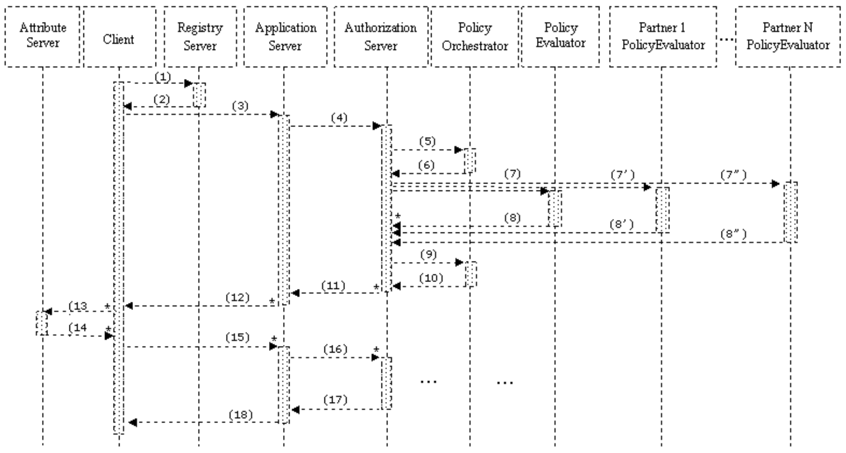


Figure 6: A data and authorization flow diagram

1. A Client asks the RegistryServer that it wants to invoke a specific service R;
2. The RegistryServer looks up for this R and returns a list of appropriate ApplicationServer(s);
3. The Client requests the ApplicationServer for invoking the service R, presenting its credentials;
4. After the ApplicationServer has received the Client's request, it checks for its AuthorizationServer offering this service (using the RegistryServer) and requests it for taking an authorization decision, presenting Client's credentials and the requested service R;
5. The AuthorizationServer queries a PolicyOrchestrator for a policy composition related to evaluating the service R;
6. The PolicyOrchestrator returns to the AuthorizationServer a graph of activities, *BPAct*, representing policy composition process;
7. The AuthorizationServer starts executing the process *BPAct* (requesting all PolicyEvaluators with respect to that *BPAct*);
8. The PolicyEvaluators return to the AuthorizationServer their access decisions either as explicit YES/NO or as a process indicating what should be done by the Client;
9. After collecting the results from all the PolicyEvaluators, the AuthorizationServer invokes a service (located at PolicyOrchestrator) indicated by *BPAct*, which is responsible for getting the final access decision based on the results from step 8 and the information release policy related to the requested service R;
10. The final access decision returned by the PolicyOrchestrator is either explicit YES/NO or a process indicating what should be done by the Client in order to get the service R;

11. The AuthorizationServer pass back to the ApplicationServer the final access decision returned by the PolicyOrchestrator;
12. The ApplicationServer enforces the access decision returned by the AuthorizationServer and sends the result back to the Client;

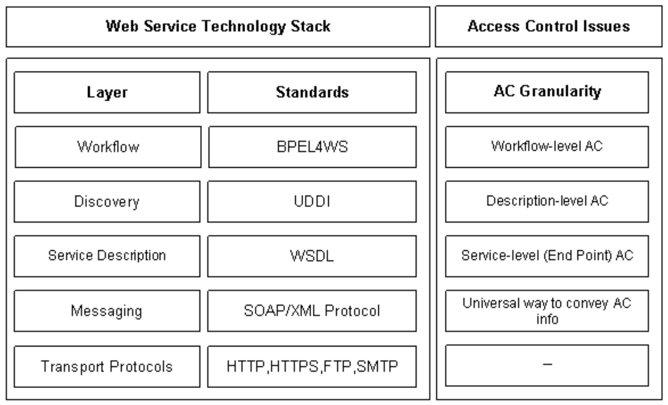
13.-15. In the case of interactive counter request returned to the Client the same starts executing it and after that re-requests the ApplicationServer for the service R presenting the new credentials it has obtained.

16. The ApplicationServer after being requested by the Client, requests the AuthorizationServer for taking the authorization decision with the new set of Client's credentials;
17. The AuthorizationServer returns the final access decision (YES/NO) to the ApplicationServer;
18. The ApplicationServer enforces the access decision returned by the AuthorizationServer and sends the result back to the Client.

B A Primer on WS and Business Processes

A Web Service as defined by the standard[9] is "an interface that describes a collection of operations that are network-accessible through standardized XML messaging. A Web service is described using a standard, formal XML notion, called its *service description*. It covers all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols and location."

The idea behind Web services is to encapsulate and make available enterprise resources in a new heterogeneous and distributed way.

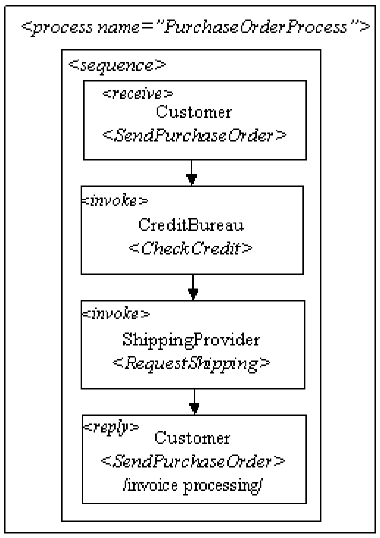


Web Service Technology Stack		Access Control Issues
Layer	**Standards**	**AC Granularity**
Workflow	BPEL4WS	Workflow-level AC
Discovery	UDDI	Description-level AC
Service Description	WSDL	Service-level (End Point) AC
Messaging	SOAP/XML Protocol	Universal way to convey AC info
Transport Protocols	HTTP,HTTPS,FTP,SMTP	–

Figure 7: Web Services Technology Stack & Access Control Issues

The WS architecture, as defined by W3C[10], is divided into five layers grouped into three main components - Wire, Description, and Discovery (Fig. 7). The *Wire* component comprises the messaging and transport layers with the SOAP protocol and the XML message format. *Discovery* offers users a unified and systematic way to find, discover, and inspect service providers over the Internet. There are

[9] Web Services Conceptual Architecture (WSCA), http://www- 3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf
[10] W3C. Web Services Architecture. http://www.w3.org/TR/ws-arch.



```
<process>
    <sequence>
        <receive partner="Customer"
                 portType="purchaseOrderPT"
                 operation="SendPurchaseOrder"
                 container="PO">
        </receive>
        <invoke  partner="CreditBureau"
                 portType="CheckCreditPT"
                 operation="CheckCredit">
        </invoke>
        <invoke partner="shippingProvider"
                portType="shippingPT"
                operation="RequestShipping"
                inputContainer="shipingRequest"
                outputContainer="shippingInfo">
                    <source linkName="ship-to-invoice">
        </invoke>
        <reply partner="Customer"
               portType="purchaseOrderPT"
               operation="SendPurchaseOrder"
               container="Invoice"/>
    </sequence>
</process>
```

Figure 8: Example of BPEL4WS Process

two standards proposed at this level - Universal Description, Discovery and Integration (UDDI) and Web Service Inspection Language (WSIL).

Moving upward we found the *Service Description* layer and the *Business Process Orchestration* layer. The service description layer is responsible for describing the basic format of offered services (protocols and encodings, where a service resides, and how to invoke it). The standard for describing the communication details at this layer is Web Service Description Language (WSDL).

The Business Process Orchestration layer is an extension of the service model defined at the description layer. This layer is responsible for describing the behavior of complex business and workflow processes. Intuitively, business processes are graphs where each node represents a business activity and primitive nodes are in WSDL. The recently released standard at this layer is the Business Process Execution Language for WS (BPEL4WS)[11].

The BPEL4WS primitive activities are the following:

`<invoke>` invoking an operation on some Web service;
`<receive>` waiting for an operation to be invoked by someone externally;
`<reply>` generating the response of an input/output operation;
`<assign>` copying data from one place to another.

More complex activities can be constructed by composition:

`<sequence>` - allows the developer to define an ordered sequence of steps;
`<switch>` - allows the developer to have branching;
`<while>` - allows the developer to define a loop;
`<flow>` - allows the developer to define that a collection of steps has to be executed in parallel.

[11]BPEL4WS specification – http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

An example of compositions of services is shown in Figure 8: a buyer service is ordering goods from a seller service, i.e. the buyer service invokes the order method on the seller service, whose interface is defined using WSDL. The seller service invokes a credit validation service to ensure that the buyer can pay for the goods and after that continue by shipping the goods to the buyer. The credit validation service can take place at a credit bureau site in a separate security domain. Notice that a number of partners participate in the process that therefore crosses administrative boundaries.

The XML code shown in Figure 8 is a very brief example of the scenario described above in the notations of BPEL4WS primitives. The structure of the processing section is defined by the `<sequence>` element, which states that the elements contained inside are executed in this order. The node contents is self explanatory.