Chapter XLIV

PolyOrBAC:
An Access Control Model for Inter–Organizational Web Services

Yves Deswarte
Université de Toulouse, LAAS-CNRS, France

Anas Abou El Kalam
Université de Toulouse, IRIT, INPT-ENSEEIHT, France

ABSTRACT

With the emergence of Web Services-based collaborative systems, new issues arise, in particular those related to security. In this context, Web Service access control should be studied, specified and enforced. This work proposes a new access control framework for Inter-Organizational Web Services: “PolyOrBAC”. On the one hand, the authors extend OrBAC (Organization-Based Access Control Model) to specify rules for intra- as well as inter-organization access control; on the other hand, they enforce these rules by applying access control mechanisms dedicated to Web Services. Furthermore, the authors propose a runtime model checker for the interactions between collaborating organizations, to verify their compliance with previously signed contracts. In this respect, not only their security framework handles secure local and remote accesses, but also deals with competition and mutual suspicion between organizations, controls the Web Service workflows and audits the different interactions. In particular, every deviation from the signed contracts triggers an alarm, the concerned parties are notified, and audits can be used as evidence for a judge to sanction the party responsible for the deviation.

1. INTRODUCTION

Web Services (WS) are increasingly gaining acceptance as a framework for facilitating application-to-application interactions within and across enterprises. In fact, WS facilitate the interoperability by providing abstractions as well as technologies for exposing enterprise applications as services and make them accessible through standardized interfaces (XML (World Wide Web
Consortium [W3C, 2004], WSDL (W3C, 2006b), SOAP (W3C, 2003)).

However, while much progress has been made toward providing interoperability, there is still a lot to do at the security level. In particular, a well-founded security study should identify who has access to what, when and in which conditions. The Common Criteria define an “organizational security policy” as: a set of security rules, procedures, or guidelines imposed (or presumed to be imposed) now and/or in the future by an actual or hypothetical organization in the operational environment (Common Criteria for Information Technology Security Evaluation, 2006a). Such an organizational security policy usually relies on an access control policy (Common Criteria for Information Technology Security Evaluation, 2006b). An access control model is often used to rigorously specify and reason on the access control policy (e.g., to verify its consistency). However, the model does not specify how the security policy is enforced. The enforcement is realized by technical security mechanisms, such as credentials, cryptographic transformations (e.g., signature, encryption), access control lists (ACL), firewall rules, etc.

Moreover, in the context of an AAA architecture, not only it is important to specify and enforce Authentication and Authorization, but it is also necessary to achieve an efficient Accounting. This is extremely important in the WS context, in particular to prove infractions and to clearly identify the responsibilities in case of dispute or abuses.

Our major aim in this chapter is to define a global framework (access control model and mechanisms) for secure WS. In our study, we give a major attention and we progressively try to satisfy the following requirements:

- **Secure cooperation** between different organizations / users offering or using WS, but possibly mutually suspicious, with different services, features, functioning rules and security policies.
- **Loosely coupled organizations**: Each organization controls (and is responsible for) its own security policy, resources, applications, etc.
- **Decentralized** enforcement and administration of the security policies: Each organization should enforce its own security policy with its own mechanisms.
- **Heterogeneity and self-determination**: As each organization is free to have its own WS, structure, OS, and local objects, it is the matter with heterogeneous systems where organizations keep some local self-determination. Actually, implementation details as well as private information should be managed by each organization, while remote accesses should be carried out through WS interfaces.
- **Fine-grained access control**: Access control decisions should take the context (e.g., specific situations, time and location constraints) into account. Moreover, as the context may change often and as certain reactivity is required in WS, organizations should support dynamic access rights.
- **Enforcement of permissions, explicit prohibitions as well as obligations**. In fact, explicit prohibitions can be particularly useful as we can have composite WS with decentralized policies where each administrator does not have details about the other parts of the system. Moreover, explicit prohibitions can also specify exceptions or limits the propagation of permissions in case of hierarchies. Similarly, obligations can be useful to impose some internal / external, manual / automatic actions that should be carried out by users or automatically performed by the system itself.
- **The security policy must be vendor- and manufacturer-independent**. Each time the
PolyOrBAC

vendor- or technology-specific statements are used, the maintenance burden for the policy increases, especially in this domain where technologies change and new acquisitions often occur. If the security policy is not updated, it becomes obsolete, which would not be acceptable in such systems. The policy must thus remain effective and as abstract as possible.

• Audit and assessment: Audit determines if the protections which are defined in the policy are correctly enforced; it also checks if the contracts / agreements established by the partner organizations and users are well-respected.

To satisfy these requirements, we suggest using the PolyOrBAC framework. The latter combines the OrBAC (Organization-Based Access Control) model (Abou El Kalam et al., 2003) with WS mechanisms. However, PolyOrBAC (enforced by traditional security mechanisms) is not completely efficient in our context. In fact, while these mechanisms are able to enforce permissions, they do not efficiently enforce obligations and explicit prohibitions, and these kinds of rules are very important for WS. Moreover, in such architectures, it is crucial to audit the different actions and interactions. As stated above, not only we should be able to keep an audit trail, but we also should precisely identify if organizations and users respect their obligations and comply with their expected behaviors. For these reasons, we present a runtime model checker (based on timed automata) that is able to verify if the different interactions are in accordance with the WS protocols.

The remainder of this chapter is organized as follows: Section 2 presents the necessary background to understand this topic as well as some important existing strategies used to secure collaborative systems. Then, the PolyOrBAC access control framework is presented in Section 3. Afterwards, Section 4 discusses our runtime model checker. Finally, in Section 5, we draw out conclusions and perspectives.

2. BACKGROUND

2.1. Standards for WS Security

In some WS implementations, security is delegated at the application level to the security mechanisms of the Web server, which acts as an application server. For example, the Tomcat Web server (Apache 2007) provides a security manager for the administration of users, groups, roles, and permissions.

On the other side, the W3C and OASIS have proposed several interesting standards for Web Services security. Let us present the most relevant ones.

The Organization for the Advancement of Structured Information Standards [OASIS] (2006) defines the Web Services Security (WSS) specification. The latter describes a mechanism for securely (by encryption and digital signature) exchanging SOAP messages (e.g., security tokens such as user name and certificates).

In 2003, OASIS defines the eXtensible Access Control Markup Language (XACML) and in 2005 it proposes the Security Assertion Markup Language (SAML). SAML is an XML-based framework for exchanging authentication data (authentication and attribute assertions) between security domains (OASIS, 2005a), while XACML is a declarative access control policy language...
implemented in XML (OASIS, 2003). XACML provides a vocabulary to specify subjects, rights, objects and conditions. Implementations of the SAML and XACML standard already exist, for example the IBM implementation (IBM, 2004), the Sun’s XACML implementation (SUN, 2005) written in Java, or the Jiffy Software (Jiffy, 2003). SAML and XACML can be used conjointly to provide interoperable policy-based access control: SAML for secure credentials exchange, and XACML for the access control policy definition.

2.2. Traditional Access Control Models for WS

Classical access control models (discretionary “DAC” and mandatory access control “MAC” (Bell & LaPadula, 1976)) are not really adapted to WS. For instance, the HRU model, defined by Harrison, Ruzzo and Ullman in 1976, represents the relationships between the subjects, the objects and the actions by a matrix $M(s, o)$. It is thus necessary to enumerate all the triples $(s, o, a)$ that correspond to permissions defined by the security policy. Moreover, when new entities are added to or removed from the system, it is necessary to update the policy.

Role Based-Access Control (RBAC) is more flexible. Roles are assigned to users, permissions are assigned to roles and users acquire permissions by playing roles (Sandhu et al., 1996; Ferraiolo et al., 2001). Hierarchical RBAC (Ahn & Sandhu, 2000) adds a requirement for supporting the role hierarchies, while Constrained RBAC enforces the separation of duties. RBAC is unquestionably suitable for a large range of organizations. Indeed, if users are added to the system, only the instances of the relationship between the users and the roles are updated.

To benefit from the advantages of RBAC, several works tried to apply it to WS. In 1991, Beznosov & Deng presented a framework for implementing Role-Based Access Control Using CORBA security service. In 2001, Vuong, Smith & Deng proposed an XML-Based approach to specify enterprise R-BAC policies; in 2004, Feng, Guoyuan & Xuzhou suggested SRBAC, a Service-oriented Role-Based Access Control model and security architecture model for Web Services; and in 2002, Leune, Van & Heuvel presented RBAC4WS, a methodology for designing and developing a Role-Based Access Control model for Web Services. Focusing at Service invocation, this methodology adopts a symmetric perspective considering both the supplier and the customer.

Besides, some other works tried to couple XACML with RBAC. For example, in 2004, OASIS adopted an XACML profile for Role Based Access Control, while in 2005, Crampton proposed an RBAC policy using an XACML formulation.

In our recent works, we enhanced the RBAC by proposing the OrBAC (Organization-based Access Control) model (Abou El Kalam et al., 2003). Basically, OrBAC is an extension of RBAC that details permissions while remaining implementation independent. The main idea is to express the security policy with abstract entities only, and thus to separate the representation of the security policy from its implementation. Indeed, OrBAC is based on roles, views, activities (introduced in RBAC, VBAC, TBAC (Sandhu et al., 1996; Brose, 1999; Thomas & Sandhu, 1997) to structure subjects, objects and actions.

In the next section, we first summarize OrBAC features and we discuss the limits of this model regarding WS requirements (Abou El Kalam et al. 2007a).

2.3. OrBAC (Organization-Based Access Control)

In OrBAC, an organization is a structured group of active entities, in which subjects play specific roles. An activity is a group of one or more actions, a view is a group of one or more objects, and a
PolyOrBAC

Figure 1. Abstracting subjects

```
    Subject   Empower   Role
       ↓       ↓         ↓
Organization
```

Figure 2. Abstracting objects

```
    Object   Use   View
       ↓       ↓         ↓
Organization
```

context is a specific situation that conditions the validity of a rule.

Actually, the Role entity is used to structure the link between the subjects and the organizations (Fig. 1). The relationship Empower (org, r, s) means that org employs subject s in role r. In the same way, the objects that satisfy a common property are specified through views (Fig. 2), and activities are used to abstract actions.

In security rules, permissions are expressed as Permission(org; r; v; a; c); obligations and prohibitions are defined similarly. Such an expression is interpreted as: in the context c, organization org grants role r the permission to perform activity a on view v.

As rules are expressed only through abstract entities, OrBAC is able to specify the security policies of several collaborating and heterogeneous organizations.

In fact, the same role, e.g., “operator” can be played by several users belonging to different organizations; the same view e.g., “TechnicalFile” can designate different objects, say TF-Table or TF1.xml, in different organizations; and the same activity “read” could correspond in a particular organization to a “SELECT” action (if the organization has a database system) while in another organization it may specify an OpenXMLfile() action.

Two security levels can be distinguished in OrBAC (Fig. 3):

- **Abstract level**: The security administrator defines security rules through abstract entities (roles, activities, views) without worrying about how each organization implements these entities.
Concrete level: When a user requests an access, concrete authorizations are granted (or not) to him according to the concerned rules, the organization, the played role, the instantiated view/activity, and the current parameters.

The derivation of permissions (i.e., instantiation of security rules) can be formally expressed as follows:

\[
\forall \text{org} \in \text{Org}, \forall \text{s} \in \text{S}, \forall \alpha \in \text{A}, \forall \text{o} \in \text{O}, \forall \text{r} \in \text{R}, \\
\forall \text{a} \in \text{A}, \forall \text{v} \in \text{V}, \forall \text{c} \in \text{C}, \\
\text{Permission } (\text{org}, \text{r}, \text{v}, \text{a}, \text{c}) \land \\
\text{Empower } (\text{org}, \text{s}, \text{r}) \land \\
\text{Consider } (\text{org}, \text{a}, \text{a}) \land \\
\text{Use } (\text{org}, \text{o}, \text{v}) \land \\
\text{Hold } (\text{org}, \text{s}, \text{a}, \text{o}, \text{c}) \\
\rightarrow \text{Is permitted}(\text{s}, \text{a}, \text{o})
\]

This rule means:

If a security rule specifies that “in org, role r can carry out the activity a on the v when the context c is True”; if “in org, r is assigned to subject s”; if “in org, action a is a part of activity a”; if “in org object o is part of view v”; and if “the context c is True for the triple (org, s, a, o)”; then s is allowed to carry out a on o.

In our context, OrBAC presents several benefits:

- **Rules expressiveness**: OrBAC defines permissions, interdictions and obligations.
- **Abstraction of the security policy**: OrBAC has a structured and an abstracted expression of the policy; it also separates the specification from the implementation of the policy.
- **Scalability**: OrBAC has no limitation in size or capacity. It can define an extensible
policy. It is then easily applicable to large-scale environments.

- **Loose coupling**: Each organization is responsible for its assets and entities. Implementation details as well as private information are managed separately by each organization.
- **Evolvability**: A policy in OrBAC is evolvable. It easily handles changes in organizations.
- **User-friendliness**: Specifying and updating an OrBAC security policy are rather intuitive.
- **Popularity**: OrBAC has a growing community. Many research studies are being conducted, based on OrBAC.

However, even if OrBAC has several advantages, is not completely adapted to WS. First, OrBAC is not able to manage collaboration-related aspects. In fact, as OrBAC security rules have the Permission(org, r, v, a, c) form, it is not possible to represent rules that involve several independent organizations, or even, autonomous sub-organizations of a particular collaborative system. Moreover, it is impossible (for the same reason) to associate permissions to users belonging to other partner-organizations (or to sub-organizations). As a result, if we can assume that OrBAC provides a framework for expressing the security policies of several organizations, it is unfortunately only adapted to centralized structures and does not cover the distribution, collaboration and interoperability needs, while these aspects are very important in the WS context.

Secondly, the translation of the security policy into access control mechanisms is not treated in OrBAC. It is thus necessary to describe suitable architecture, scenario and implementation of the WS security.

To cover these limitations, we propose the PolyOrBAC framework.

3. POLYORBAC

In this section, we suggest adapting OrBAC as well as WS mechanisms to specify and enforce secure collaboration (Abou El Kalam et al., 2007a), in particular in the WS context. The global framework is called PolyOrBAC. The main idea is:

- Extending OrBAC to be able to express collaboration rules concerning remote accesses.

- Using existing WS standards to enforce the collaboration at service and resource levels.

3.1. Scenario of Execution

Let us develop a simplified (but representative) scenario illustrating collaborative systems in general and WS in particular. We distinguish two global phases.

**First phase**: Publication and negotiation of collaboration rules as well as the corresponding access control rules.

First, each organization determines which resources it will offer to external partners. Web services are then developed on application servers, and referenced on the Web Interface (in UDDI (OASIS, 2005b)) to be accessible to external users. At this stage, we find in organization B security rules such as: Permission(B, Accountant, Account, Consulting, Urgency) and instances (of relations) such as:

- **Empower**(B, Bob, Accountant),
- **Consider**(B, OpenXMLFile(), Consulting),
- **Use**(B, WS1, Account).

Second, when an organization publishes its WS at the UDDI registry, the other organizations can contact it to express their intent to collaborate and use the published services under an agreed access control policy. In the example below, organization
B offers WS1, and organization A is interested in using WS1.

Third, organizations A and B negotiate and come to an agreement concerning the use of WS1.

Fourth, A and B establish a contract and jointly define security rules concerning the access to WS1. These security rules are registered – according to an OrBAC format – in a database containing the policy. The steps of this phase are given in Figure 4.

For example, if the agreement between A and B is “users from A have the permission to consult B’s accounts”, B should add the Empower(B, PartnerA, Accountant) association to its base. In this notation, PartnerA means any user from A.

We assume that the security policy database already contains the rule Permission(B, Accountant, Account, Consulting, Urgency).

The derivation of the permission (i.e., instantiation of security rules) mentioned above can be formally expressed as shown (Fig. 5).

**Second phase:** Access to remote services.

At runtime, if a user wants to carry out an activity, the security-related services check the requestor/request authentication, verify its credentials, make an authorization decision based on the security policy, and finally, deny or authorize the access (in some cases, this access is accompanied with some obligations or recommendations). In this

---

**Figure 4. Mutual negotiation of access rules for distant services**

**Figure 5. Derivation of permissions in PolyOrBAC**

\[\text{Permission}(B, \text{Accountant}, \text{Account}, \text{Consulting}, \text{Urgency}) \land \text{Empower}(B, \text{Bob}, \text{Accountant}) \land \text{Consider}(B, \text{OpenXMLFile()}, \text{Consulting}) \land \text{Use}(B, \text{WS1}, \text{Account}) \land \text{Hold}(B, \text{PartnerA}, \text{OpenXMLFile()}, \text{WS1, Urgency}) \rightarrow \text{Is permitted}(\text{PartnerA, OpenXMLFile()}, \text{WS1})\]
vision, it is important to separate authentication from authorization, and access control decision from access control enforcement.

In our study, we use an AAA (*Authentication, Authorization, and Accounting*) architecture: the authorization decision is asked for by a requestor (user) or a resource service; if the security policy allows this access, an authorization ticket is delivered to the requestor; the latter presents the ticket with the authorization context to the resource or service, which enforces the decision, i.e., grants or denies the access. More precisely, if a user from “A” (let us note it *Alice*) wants to carry out an activity, *Alice* is first authenticated. Then, the access control decision function of organization *A* checks if the OrBAC security policy of *A* allows this activity. We suppose that this activity contains local as well as external accesses. Local accesses should be controlled according to *A*’s security policy, while external accesses should respect the agreements established between organization “A” and the other organizations (providing the requested services).

If, for example, *Alice*’s activity invokes (among others) *B*’s Web service *WS1*, the access to *WS1* should be controlled by *B*’s Policy Decision Point (“PDP”), according to: (1) The OrBAC policy of *B*, and (2) the agreement established between *A* and *B* about *WS1*.

It is important to note that the same (abstract) rule, e.g., *Permission*(*B, Accountant, Account, Consulting, Urgency*), can correspond to local as well as collaborative (remote) accesses. The decision corresponding to local access can be done according to:

\[
\text{Permission}(B, \text{Accountant}, \text{Account}, \text{Consulting, Urgency}) \land
\text{Empower}(B, \text{Bob, Accountant}) \land
\text{Consider}(B, \text{SELECT, Consulting}) \land
\text{Use}(B, \text{Table1, Account}) \land
\text{Hold}(B, \text{Bob, SELECT, Table1, Urgency}) \rightarrow \text{Is permitted}(\text{Bob, SELECT, Table1}),
\]

while the decision corresponding to remote access can be done according to the rule presented in Figure 5.

Let us also remind that the decision of “Which user from *A* is associated to *PartnerA*, and so, authorized to access to *WS1*” is done according to *A*’s security policy. In other words, *A* defines internally instances such as (*Alice, PartnerA*), (*Jean, PartnerA*).

In this way, when *Alice* is authenticated and authorized (by *A*’s policy) to play the role *PartnerA*, an XML-based authorization ticket “*T1*” is generated (based on the positive decision) and granted to *Alice*.

*T1* contains the following elements:

- The virtual user played by *Alice*: “*PartnerA*”,
- *Alice*’s organization: “*A*”,
- The agreement’s (between *A* and *B*) ID,
- The requested service: “*WS1*”,
- The invoked method, e.g., “OpenXML-File()”,
- And, a timestamp to prevent reply attacks.

Note that *T1* is delivered to any user (from *A*) allowed to access to *WS1* (e.g., *Jean*). When *Alice* presents its request as well as *T1* (as a proof) to *B*, *B* extracts the *T1*’s parameters, and processes the request. By consulting its security rules, *B* associates the role Accountant to the virtual user “*PartnerA*” (representing *Alice* in *B*) according to *Empower*(*B, PartnerA, Accountant*). The access decision is then done according to the rule presented in Figure 5.

### 3.2. WS Mechanisms in PolyOrBAC

In our implementation, as we use a WS-based architecture, messages exchanged (e.g., services) between *A* and *B* are XML files that obey SOAP protocols. Moreover, PolyOrBAC could be integrated perfectly into XACML architecture (Figure 6).
In this architecture, an access request arrives at the Policy Enforcement Point (PEP), the PEP creates an XACML request and sends it to the Policy Decision Point (PDP), which evaluates the request and sends back a response. The response can be either access permitted or denied, with the appropriate obligations. The PDP comes to a decision after evaluating the relevant policies. To get the policies, the PDP uses the PAP to extract the security rules (e.g., Permission(Organization, Role, View, Activity, Context)). The PDP may also invoke the Policy Information Point (PIP) service to retrieve the attribute values related to the organization, the subject, the Web service (resource), or the environment (the context). This consists in evaluating the associations Empower (org, s, r), Consider (org, α, a), Use (org, o, v) and Hold (org, s, α, o, c). The authorization decision arrived at by the PDP is sent to the PEP. The PEP:

- Fulfills the obligations and/or informs the subject about the recommendations, and,
- Based on the authorization decision sent by PDP, either permits or denies access.

Figure 7 describes the components of a PolyOrBAC implementation based on XACML (the target contains instances of (Organization, Role, View, Activity)).

Our work can perfectly be coupled with other interesting XML-based standards such as SAML, and WS-Security.

SAML could be used for authentication. It allows a user to log-on once when accessing to different sites (e.g., single sign on). SAML handles the exchanges of requests and responses containing authentication, authorization, and non-repudiation data. Those exchanges are done between the authority asserting that a certain user can use a specific WS, the user that wants to use the WS, and the entity that offers the WS.

### 3.3. Discussion

PolyOrBAC offers several benefits:

- **Peer to peer approach**: We use a decentralized architecture where organizations and users mutually negotiate their common rules; each organization is responsible for its user’s authentication and is liable for their use of other organizations’ services; it also controls the access to its own resources and services.
• **Independence:** Even if all PolyOrBAC rules are specified according to OrBAC, organizations are loosely coupled, e.g., each organization keeps its specific security policy, security objectives, services, applications, operating system, etc.

• **Information confidentiality:** Accesses are only possible through the WS interface; in this way communications are possible without intimate knowledge of the organization’s IT systems. Such information is usually considered as confidential by organizations.

• **Extensible structure:** The OrBAC extensibility and the WS standards facilitate the management and the integration of new organizations (with their users, data, services, policy, etc.).

However, PolyOrBAC has some limitations, essentially related to the last requirement identified in the introduction. In fact, PolyOrBAC:

• Offers the possibility to grant local accesses to other organizations’ users that play certain roles (in their organizations), without having any information about who plays these roles and how the (user, role) association is managed by these organizations;

• Supports access control enforcement and real time checking of contracts established between different organizations and users; in fact, the system must be able to check the well-respect of the signed contracts, according to the WS exchange protocol;

• Supports audit logging and assessment of the different actions; every deviation from the signed contracts should trigger an alarm and notify the concerned parties.

Actually, as stated above, when an organization publishes its WS at the UDDI registry, the other organizations can contact it to express their intent to use the published services under an agreed access control policy. This policy is translated into an e-contract; the latter will handle the use of this WS.

In this work we extend timed automata, initially defined in 1994 by Dill & Alur, to capture e-contracts security requirements and to verify some security properties by using a runtime model.
checking of the WS exchange protocol (Abou El Kalam & Deswarte, 2008).

4. RUNTIME MODEL CHECKING OF WS EXchanged MESSAGES

4.1. General Security Requirements

Let us start this section by presenting an example of a WS contract that stipulates the interactions between a buyer and a seller for the purchase of goods. The contract contains clauses of the following form:

- At his discretion, the buyer may send a purchase order to the seller.
- The seller is obliged to confirm acceptance/rejection of the purchase order within 24 hrs.
- The seller is obliged to send an invoice to the buyer within 7 days of accepting the purchase order.
- The buyer and the seller are forbidden to send invalid messages.
- Failures to honor obligations and prohibitions will result in financial compensations equal to 20% of the value of the item.

If one of the contractual parties detects a technical failure that prevents them from continuing the normal course of a transaction, this party is obliged to send a failure notification message to the other party as well as to the administrator/third party.

In case of failure, the e-contract is terminated and the involved parties are informed.

As we can see in this example, a WS contract should model the following requirements and entities (at least):

- **Actors:** Who are the actors (organizations, roles) involved in the contract? Who can carry out actions according to the contract clauses?
- **Heterogeneity:** WS contracts may have complex structures based on bilateral (e.g., buyer-seller), or multiparty (e.g., house building contract) interactions; this naturally implies different views, structures, and implementations.
- **Actions/workflows:** As the contract should identify the activities (tasks/e-services to be executed during the WS protocol), the e-contract security policy should handle the concepts of actions and workflows (sequential, cyclic, ordered, …).
- **Deontic modalities:** A WS contract can be seen as a legally enforceable agreement in which two or more parties commit to certain obligations. Consequently, classical deontic concepts, such as obligations, permissions and prohibitions are important in modeling the contract security policy.
- **Temporal modalities:** Temporal constraints are rules that regulate the order, timing and duration of actions. It is thus obvious that WS contracts should identify the duration (e.g., number of days/hours) of certain actions, the synchronization requirements as well as when (at, before, after, during …) an action can/must/cannot be carried out.
- **Context:** In order to provide fine-grained access control, it is necessary to take the context into account, e.g., the requestor/resource location, the separation of duty, delegations, exceptions.
- **Sanctions:** As WS contracts contain deontic modalities (e.g., obligations, prohibitions), it is necessary to handle the cases where these modalities are not respected. In particular, a failure to execute an obligation and an attempt to execute a prohibition are considered as contract violations and the offending actor may be subject to sanctions.
- **Auditing:** When implementing an e-contract tool, it is necessary to keep an audit log that
PolyOrBAC can be analyzed at runtime and/or be used later as evidence in case of disputes.

4.2. A Security Model for a WS Contracts

The previous section presented the most relevant entities and requirements (mainly related to temporal constraints, actions / workflows, deontic modalities and sanctions) that should be handled by an e-contract security policy. The challenge now is to find a convenient framework that captures all these aspects. Actually, we believe that most of these requirements (except deontic modalities) can be specified by timed automata. Our choice is also motivated by the possibility of checking the correctness of the automata behavior and by the availability of several tools dedicated to this issue.

In this context, our methodology consists in:
- (1) showing how timed automata can capture some WS contract security requirements;
- (2) trying to homogeneously extend timed automata to capture the deontic modalities (Abou El Kalam & Deswarte, 2008). But before explaining these steps, let us first present the most relevant notions (regarding our study) of timed automata.

4.2.1. Timed Automata

Timed automata have been proposed by Alur and Dill to describe systems behavior with time (Alur 1994). Basically, a timed automaton is a finite automaton with a set of clocks, i.e., real and positive variables increasing uniformly with time. Transition labels are: a guard, i.e., a condition on clock values, actions and updates, which assign new values to clocks.

Composition of timed automata is obtained by synchronous product: each action $a$ executed by a timed automaton corresponds to an action with the same name $a$ executed in parallel by a second timed automaton. In other words, a transition that executes the action $a$ can only be triggered in one automaton if the transition labeled $a$ can also be triggered in the other automaton. The two transitions are preformed simultaneously and communications use the rendez-vous mechanism. Note that performing transitions is instantaneous; conversely, time can be consumed in nodes.

Besides, each node is labeled by an invariant, that is a Boolean condition on clocks. Node occupation is invariant-dependent. The node is occupied if the invariant is true.

Finally, it is important to note that a system modeled with timed automata can be verified using model-checking. In particular, a reachability analysis can be performed by model-checking. It consists in encoding a certain property in terms of reachability of a given node (of one of the automata). In this respect, the property is verified by a node reachability if and only if the node is reachable from an initial configuration.

In the following subsections, we model the different requirements. The properties will be verified using the UPPAAL model-checker (UPPAAL 2006; Larsen and Pettersson, 1997).

4.2.2. Modeling Permissions

Permissions mean actions that are allowed by the contract’ clauses. In our timed automata model, permitted actions are actually specified by transitions. For instance, in Figure 8, the system can execute the action $a$ at any time and then, behaves like the automaton $A$.

4.2.3. Modeling Prohibitions

We distinguish two kinds of prohibitions:

- **Implicit prohibitions**: The idea is to only specify permissions, which means that prohibited actions (i.e., actions that are not in accordance with the contract clauses) do not need to be specified explicitly in the automata. The states, actions and transitions not represented in the automata are
Figure 8. Permissions modeling

![Permission Diagram](image)

Figure 9. Modeling prohibitions

![Prohibition Diagram](image)

by essence not possible because the system will not recognize them. This policy is actually similar to “by default-policies” used in some firewall configurations (i.e., in the context of such firewalls, actions that are not explicitly specified as permitted are actually prohibited).

- **Explicit prohibitions**: Explicit prohibitions can be particularly useful in the management of decentralized policies/contracts where each administrator does not have details about the other parts of the system. Moreover, explicit prohibitions can also specify exceptions or limit the propagation of permissions in case of hierarchies. In our model, we specify explicit prohibitions by adding a “failure state” where the user will be (automatically) leaded if he/she misuses the system (Figure 2).

In Figure 9, as the $a$ action is forbidden, its execution automatically leads to the failure state described by an unhappy face.

### 4.2.4. Modeling Obligations

Obligations are actions that must be carried out; otherwise the concerned entity will be subject to sanctions. In particular, obligations can be useful to impose some internal or external, manual or automatic actions.

In this work, we distinguish two kinds of obligations: *internal* obligations and *external* obligations.

- An *internal obligation* is a set of mandatory actions that must be performed by local entities (possibly constrained by a maximum delay). An obligation is automatically trig-
triggered by an event such as a change in the context or a particular message exchanged between the contractual entities (e.g., the buyer and the seller).

- An **external obligation** is a set of mandatory actions that **must be performed by remote entities**, but **are checked by local entities**.

Note that the difference between these two kinds of obligations is related to the target of the action (i.e., the entity subject to the obligation), not to the way the action is carried out.

This distinction is important in the context of e-contracts as each contractual party has its own automaton, which checks/verifies if the behavior of the other party respects the contract clauses. In fact, in each automaton, it would be important to distinguish between actions that must be done internally and those that must be performed by the other party.

After defining obligations, let us now explain how we model these notions with timed automata. First, as every obligation is also a permission (Obligation → Permission), obligations will be specified by particular transitions (in the same way as permissions); however, as obligations are stronger than permissions, we should add other symbols to capture this semantic and to distinguish between what is mandatory and what is permitted but not mandatory.

Actually, to model obligations, we use transition time-outs and invariants.

In this respect, an internal obligation is considered as a simple transition, and if a maximum delay is assigned to the obligation, a time-out (noted by \( d \) in Figure 10) is set for the delay. When the obligation is fulfilled by local entities, this event resets the time-out and the system behaves like \( A_1 \). On the contrary, if the time-out expires, an exception is raised, which is another internal obligation, and the system behaves like \( A_2 \) (which can be considered as an exception).

Besides, external obligations are represented by a transition with a time-out (an alternative timed transition). When the remote entity has fulfilled its obligation, it sends a message carrying a proof of obligation completion, which resets the time-out. Alternatively, if the time-out expires, an internal obligation is triggered, corresponding to an exception processing that must be performed by local entities.

In Figure 11, the two automata are running in parallel and should be synchronized on the \( a \) action. Actually, \( a \) is an external obligation that must be carried out by the right hand automaton of Figure 4. Besides, the left hand automaton should receive a signal proving the correct-execution of \( a \)

---

*Figure 10. Modeling internal obligations*
4.2.5. Representing Disputes in Our Model

Naturally, a conflicting situation may arise if one of the contract parties does not comply with the contract clauses (e.g., fails to fulfill one of its obligations, or performs a prohibited action). Of course, these dispute situations should be described in the e-contract in order to rigorously state how to solve them and which sanctions to apply. Moreover, each contractual party should maintain an audit log in order to be able to present evidences to a judge in case of dispute. According to these evidences and to the contract clauses, the judge will decide and enforce the corresponding / suitable sanctions.

To model these notions, we use two notions: a dispute state (noted by unhappy face state an in Fig. 12) and variables. Basically, when the conflicting situation is detected by one of the contracting parties, the automata automatically makes a transition to a dispute situation (i.e., to the unhappy state). Contrarily to prohibitions, disputes are stronger (more sensitive) and automatically lead to the end of the contract.

Moreover, as disputes have different severities / sensitivities and as they are not all subject to the same sanctions, we use variables (i.e., labels on the unhappy state) to distinguish the different kinds of disputes as well as the corresponding sanctions.
In this section, we have presented an homogeneous model to specify the most relevant security requirements for e-contracts (workflows, actions, permissions, prohibitions, obligations, time constraints, disputes and sanctions).

In the next section, we present our implementation as well as the mechanisms we use to verify properties such as reachability and correctness.

### 4.3. The Verification Process

Once we have defined the timed automata of our contract, it would be interesting to check its correct-execution.

Actually, the idea is to verify that the automata will never reach the dispute state. In timed automata, the reachability analysis can be carried out by model-checking. Basically, we should first specify the reachability property (e.g., reaching the unhappy state); then, we can verify if from the initial configuration, there exists a possible execution of the system that leads to a node where this property is verified (i.e., reaching the unhappy state). In this respect, the contract is satisfied if none of the possible executions of the system will lead to the unhappy state (specified through the reachability property).

Note that the reachability problem is decidable and that the reachability properties are generally simple to verify by model checkers (Bérard 2001).

### 4.4. Example of Contract Modeled by Timed Automata-Based Model

After defining our templates as well as our verification process, let us apply our model to the use case presented in Section 4.1.

In the timed automaton of Figure 13, the first transition specifies that the buyer may (has the permission to) send a purchase order to the seller. When he does it, he reaches a state where he is waiting for the accept signal from the buyer. Actually this signal corresponds to an external obligation that must be fulfilled by the seller and checked by the buyer automaton. Consequently, we represent it by a transition with a time-out initialized to 1 (1 day = 24 hours) as the seller is obliged to confirm

---

**Figure 13. The buyer e-contract model**

---
the acceptance / rejection within 1 day. When the remote entity (seller) has fulfilled its obligation (sending the acceptance within 1 day), it sends a message carrying a proof of obligation completion, which resets the time-out. Alternatively, if the time-out expires, an exception is triggered, leading to the dispute state.

Similarly, an external obligation for the remote entity (i.e., the seller) to send an invoice within 7 days (after a confirmed order) can be represented by a time-out (set for a 7 day duration). If the message corresponding to the invoice is received, the time-out is reset. On the contrary, if the time-out expires, it triggers an exception (dispute state: the e-contract is canceled, and the buyer informs a judge).

Beside that, in the timed automaton of Figure 14, the seller receives the purchase order from the buyer. Actually, it is the matter of a synchronization action (see also Figure 13). After receiving this signal it reaches a state with a “true” invariant. This means that it can reach and stay in this state without condition. From this state, three transitions are possible. If he/she accepts it within 1 day, or if he/she rejects it within 1 day, he/she fulfills its obligation. Else, he/she will receive a signal (non conformance to the obligation) from the buyer, leading the system to the dispute state.

The same reasoning goes for the rest of the automaton (the obligation to send an invoice within 7 days).

Now, once we have modeled the contractual parties’ behavior, we can verify if the system can reach a dispute state. In fact, proving that all the possible executions of the system will never lead to a conflicting situation is equivalent to prove that the exchange protocol can be run according to the contract clauses.

In our implementation, the automata are modeled by the UPPAAL model checker (UPPAAL 2008; Larsen and Pettersson, 1997). The reachability properties are modeled by a subset of the Computational Tree Logic (CTL). For example, to identify the possible executions of the system where the buyer will reach a dispute state (at list once), we can specify and trace the following property:

Figure 14. The seller e-contract model
In this property, “E” stands for *it exists* and “<>” means *at least one execution*. Thanks to the model checker (UPPAAL in our case), we have obtained two possible executions:

- The seller does not respect the acceptance delay (1 day);
- The seller does not respect the invoice delay (7 days);

Inversely, the following property means that none of the possible executions will lead the buyer to a dispute state.

\[ \text{A}[] \text{not Buyer.Dispute} \]

### 4.5. Related Works

Several works have been devoted to logics and theories for e-contracts. The most relevant ones are based on predicate logic, first order logic and speech act theory, deontic logic, model action logic, temporal logic, Petri nets and event calculus.

In the SeCo project, Gisler et al. have presented secure e-contracts based on three levels: logic, information and communication levels (Gisler et al., 1999).

Jimenez et al. have defined a method for specifying contract mediated interactions (Jimenez et al., 2005). Their model is based on deontic modalities. The general form of a contract is: \( n_s : \delta \rightarrow \theta s,b (\alpha<\psi) \), where \( n_s \) ( \( i \geq 1 \) ) is a label that identifies the \( i \)th normative statement of the contract; \( \delta \) stands for a *condition* that might eventually become true; \( \theta \) stands for *permission, obligation or prohibition*; \( s \) and \( b \) are the *subject* and *beneficiary* of \( \theta \), respectively; \( \alpha \) is an *action* to be performed by \( s \) for the benefit of \( b \); \( \psi \) is a *deadline*.

O.Perrin and Godard presented in (Perrin and Godard, 2004) an approach (somewhat similar to (Jimenez et al., 2005)) where permissions, obligations and prohibitions are mapped into ECA (even-condition-actions). An executable contract becomes a set of ECA rules deployed within a trusted third party and placed between two business partners to drive their interactions.

Kumar et al. (2007) proposed an interesting work that couple RBAC (Role-Based Access Control) with TBAC (Task-Based Access Control) and enforces sequential and temporal constraints over them so that process participants get only “Need to know information” with less administrative overhead.

Another interesting work is described by Marjanovic and Milosevic (2001). They *present a specification of deontic constraints and verification of deontic consistency associated with roles in a contract. Their model defines also temporal constraints. In their model, a duration constraint has the Duration\((ai, <=, d, b)\) form: action \( ai \) must be completed in no more than \( d \) time. Concerning the deontic modalities, they give examples such as O\((R_1, ai, e, <=, Date, t_1, t_2)\) role \( R_1 \) is obliged to finish action \( ai \) no later than \( Date \). This obligation is valid from time \( t_1 \) to \( t_2 \).

However, up to our knowledge, most of these works have combined several logics to gain in the expressiveness, while these logics / models are not necessarily homogeneous. As a result, the verification mechanisms are either rarely presented or often too complex.

Moreover, several other works based on deontic logic are not expressive enough to describe situations where neither actions are assigned to specific agents (“*it ought to be that the payment is sent by Alice*”), nor permissions, obligations and prohibitions that become and cease to be in effect depending on the occurrence of time and other events (deontic logic is actually static).

To overcome these limits, some works mix deontic logic with constructs from Modal Logic, Temporal Logic, Logic of Action or from their combinations. Such hybrid logic systems can certainly express complex situations; however
such systems have not yet been thoroughly studied and understood. Moreover, the logical rigor of a contract expressed in such notations is questionable; and generally, the resulting language is not really able to automatically verify the correctness of his notation by proof-theoretical means or model checking.

In our work, we based our model on well-known mechanisms ans and tools: timed automata, CTL and UPPAAL. To be as rich as possible while remaining verification-homogeneous, we gave a precise semantic to specify and distinguish between the different deontic modalities (permissions, obligations, prohibitions), the dispute situations and sanctions.

CONCLUSION AND PERSPECTIVES

This chapter presented an access control framework for WS. We first identified the most relevant security-related requirements of WS. Then, according to these requirements, we discussed related works as well as existing WS security-related standards. Afterwards, we proposed the PolyOrBAC access control model and we emphasized its benefits and limits. In particular, PolyOrBAC offers a decentralized management of the access control policies and an architecture where organizations are independent and maintain independently their specific security policy, security objectives, services, applications, operating system, etc.; rights / permissions are defined and enforced by each organization security mechanisms; accesses are only possible through the WS interface and internal details of each organization are not visible to external users.

Then, we enhanced PolyOrBAC by a runtime model checking framework that captures the security requirements of WS contracts and that can be instantiated according to the actual context of a given WS. Our model checker is also used to monitor the well execution of the WS contracts and to verify some security properties.

Currently, we are looking for enhancing our model to take into account other interesting requirements such as delegations. Furthermore, liability as well as privacy issues are also serious challenges that need to be addressed in this field.

ACKNOWLEDGMENT

This study is partially supported by the European FP6-IST Project CRUTIAL (CRitical Utility Infrastuctural Resilience), the European Network of Excellence ReSIST (Resilience for survivability in IST), the LAAS Project PolSec (Politiques de Sécurité) and the ADCN2 Airbus project.

REFERENCES


**KEY TERMS AND DEFINITIONS**

*Availability* is the readiness for correct service when needed by authorized users.
**Collaboration** refers to all processes wherein people (or machines, or applications) work together - applying both to the work of individuals as well as larger collectives and societies.

**Confidentiality** is the absence of unauthorized disclosure of information or functions. It implies that information is readable only to authorized users.

**Integrity** is the absence of improper system state alterations. It implies that data is modified only by authorized users and only in an authorized manner.

**Interoperability** is the ability of products, systems, or business processes to work together to accomplish a common task.

**Model Checking** is a formal verification technique that compares the implementation of a design to a set of user-specified properties. It determines whether a set of properties hold true for the given implementation of a design.

**Security** studies problems, methods and solutions related to the three security properties: availability, confidentiality, and integrity.

**Security Mechanisms** are techniques used to implement the authentication and authorization, e.g., credentials, capacities, cryptographic transformations such as signature and encryption, access control lists (ACL).

**Security Model** rigorously defines a security policy. Generally, a security model is a “formal system” used to specify and reason on the security policy (i.e., it is used as a basis for formal specification proofs). It is thus intended to abstract the security policy and handle its complexity; represent the secure states of a system as well as the way in which the system may evolve, verify the consistency of the security policy, detect and resolve possible conflicts.

**Security Policy** is the set of laws, rules, and practices that regulate how an organization manages, protects, and distributes sensitive information. Basically, a security policy is specified through: the security objectives that must be satisfied (expressed in terms of confidentiality, integrity and availability) and the security rules expressing how the system may evolve in a secure way (who has access to what and in which conditions).