

Security Architecture for Virtual Organizations of Business Web Services

Florian Kerschbaum *

SAP Research, Karlsruhe, Germany

Philip Robinson

SAP Research, Belfast, United Kingdom

Abstract

Virtual Organizations (VO) temporarily aggregate resources of different domains to achieve a common goal. Web services are being positioned as the technological framework for achieving this aggregation in the context of cross-organizational business applications. Numerous architectures have been proposed for securing VOs, mostly for scientific research, such that they do not address all the requirements of business-oriented applications. This paper describes these additional requirements and proposes a novel architecture and approach to managing VO access control policies. Business users can focus on designing business processes, exposing web services and managing their VO partnerships, while the architecture supports and secures the web service interactions involved.

Key words: Virtual Organization, Virtual Organization Management Security, Web Service Security, Access Control Derivation

1 Introduction

A *Virtual Organization (VO)*, as described by Foster, Kesselman and Tuecke [7], comprises of a highly-distributed, cross-domain computing infrastructure that is dedicated to supporting large-scale resource sharing, resulting in dynamic

* Corresponding author.

SAP Research, Vincenz-Prießnitz-Str. 1, 76131 Karlsruhe, Germany

Tel.: +49 6227 7 52577 Fax: +49 6227 78 45465

Email addresses: florian.kerschbaum@sap.com (Florian Kerschbaum),
philip.robinson@sap.com (Philip Robinson).

collection of *resource providers*. VOs are considered to be part of the solution to what is termed as the "Grid Problem" [7], where flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, [businesses] and their resources is required. The goals include increasing productivity, quality of products and services and response to changes in the market. With the maturity and standardization of Web-Service (WS) Technologies, the business world envisions dynamically configured and connected WS endpoints belonging to different resource providers, which can be autonomously managed by their respective providers and "reset" once the usage of the interconnection is no longer required. We refer to this vision as *Virtual Organizations of Business Web Services*. Although the vision is relatively easy to conceive, comprehensive solutions to configuring and managing these interconnected solutions in a secure manner while being dynamic are still not yet available. We refer to this as the security management problem for VOs of business web services, stated simply as the *security management problem*. The distributed, cross-domain nature of VOs serves to make solutions to the security management problem more of a challenge. In order to solve this problem comprehensively, the following conceptual and technical requirements are proposed:

- (1) **Automation of the access control and key management processes** for each participant in the VO, such that access controls are enabled and disabled with a *least privileges property*. A system is said to maintain a least privileges property if and only if for all positive authorizations enabled in the system, there exists at least one task, responsibility or directive to be fulfilled that necessitates the existence and enabling of the respective access control.
- (2) **Autonomy of participating organizations in the VO** must be maintained. Participation in a VO should include the signing of contractual agreements and accepting of responsibilities without forcing participants to relinquish the control of their resources, firewall rules, web services, internal policies and internal organization of roles. Delegating the specification, distribution, enforcement and state management of access controls and keys to a central authority or VO manager should be avoided.
- (3) A **minimal amount of networked/remote procedure calls** should be required for distributed access control and key management in the VO. Juric et al. [11] show that WS-Security messages are 6.9 times larger than standard WS SOAP messages and introduce a large latency factor for even basic or string data types.
- (4) A **minimal trusted computing base** should be aimed for as the VO and security (access control and key) management solution implemented by each participant. The amount of new trusted software that needs to be either installed at each participant or interconnected with (e.g. via web service interfaces) should be kept to a minimum. The larger the trusted computing base the greater the likelihood of bugs and vulnerabilities

being discovered by attackers.

- (5) Along with the minimal trusted computing base, a **single, comprehensive security solution that protects both business and VO management web services** is required. Many VO security solutions focus only on the protection of resources and services offered in the VO but treat the protection of the resources and services required for life-cycle and membership management as a separate issue.

The paper proceeds to describe related work towards addressing these requirements in Section 2, proposes an architecture in Section 3 and describes two main features of the architecture in detail in Sections 4 and 5 respectively: *control-flow aware authorization policy derivation*, which builds on previous work in [20]) and *secure VO management*, which builds on previous work in [5,14].

2 Related Work

This section describes background work towards solving the security management problem for VOs of Business Web Services. In doing so, related work in the areas of (1) VO security management and (2) business process security are considered.

2.1 VO Security Management

Approaches to VO security management are either *capability-issuing* or *policy-issuing*. These are represented by subfigures (A) and (B) in Figure 1 respectively, showing trust relationships and interactions. Authorization policies and capabilities have the same general structure and syntax but different execution semantics. Authorization policies and capabilities are defined as a triple $\langle \text{subject } s, \text{object } o, \text{action } a \rangle$, stating that a subject s can perform action a on object o [8,21]. In the case of authorization policies, incoming messages are only handled if they contain a subject and target that match the policy. In the case of capabilities, the policy is carried with the message as an assertion, stating to the resource provider that the subject is authorized to perform the operation on the target.

In Figure 1, both cases (A) and (B) assume a trusted credential service (e.g. a certificate authority) for the purpose of verifying identities and attributes of clients. Note that the terms resource and web service are used interchangeably within this paper, in order to switch between specifying abstract concepts and explaining concrete applications of these concepts.

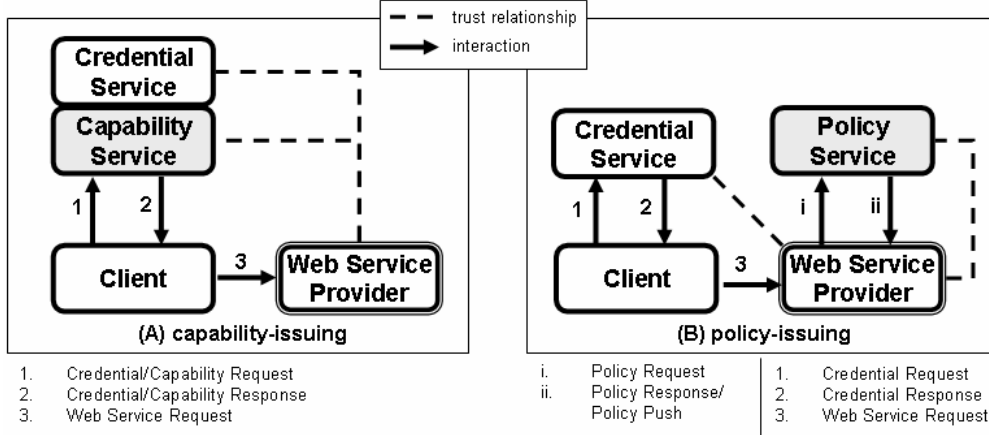


Fig. 1. Two different general approaches to VO security management

These two classes of VO security architectures are discussed below, using examples of established solutions.

2.1.1 Capability-issuing VO security architecture

These architectures feature a trusted *capability service* that issues signed assertions of the form $\langle s, o, a \rangle$ to clients, which the clients then include in the headers of their web service interactions. Two well-known examples are CAS (Community Authorization Service) [19], the security architecture for Globus and VOMS (Virtual Organization Membership Service) [1,6], a similar solution to CAS developed in the EDG (European Data Grid) project. The two solutions differ with respect to the format of the capabilities and the granularity of requests made by clients issued with the capabilities. Secondly, in the case of CAS [19], each resource provider contributes blocks of their resources to a "community" and delegates the issuing of access capabilities to the CAS server. VOMS maintains a so-called gridmap-file that represents an extended access control list with groups and roles. Both CAS and VOMS however implement the same basic capability-issuing protocol, as shown in Figure 1-(A) and described below:

1. *Capability Request*: a client establishes a secure channel with a capability service and issues a request for a signed token asserting permission to perform an operation on a target resource.
2. *Capability Response*: if the capability service determines (using its internal policy decision process) that the client's request is valid, the capability service returns a signed token with the structure $\langle s, o, a \rangle$, representing the issued capability.
3. *Web Service Request*: the client then issues a request to a target web service, including the capability in the message's header.

Access control decisions are made by a resource provider by inspecting the permissions asserted in the capability. In the case of CAS [19] capabilities are typically encoded as SAML (Security Assertion Markup Language) [4] tokens, while VOMS' capabilities are extended X.509 certificates. Resource providers trust that the capability service (CAS or VOMS) has issued the capabilities to the client for a legitimate purpose in the VO. The credential service and capability service often have the same provider, as capabilities are often embedded in credentials. In such a setting the resource provider only needs to install the trusted public key certificate of the credential service in order to technically assert a trust relationship.

2.1.2 Policy-issuing VO security architectures

These architectures feature a trusted *policy service* that issues signed policies of the form $\langle s, o, a \rangle$ to resource providers. Akenti [10] is the most established example of this form of VO security architecture, where policies are issued and encoded in the form of X.509 certificates extensions. Policies are therefore referred to as *policy certificates*, which also have an expiry or other means of revocation similar to more traditional identity certificates. The policy service and credential service may have the same provider but policies are not embedded in credentials, such that they still act as two separate services. There are then two protocols managed by the resource provider, as shown in Figure 1-(B):

- i. *Policy Request*: a resource provider may request policy updates from a policy service based on expiration or an event that a new VO is being created.
- ii. *Policy Response/ Policy Push*: the policy service creates and signs access control policies that are relevant to the resource provider (i.e. where the targets in the policies are resources belonging to the resource provider).
 1. *Credential Request*: a client requests credentials that match the subject of the policies it assumes to be installed by the resource provider of the targeted web service.
 2. *Credential Response*: a credential response provides a token that proves the client's subject identity, roles and attributes.
 3. *Web Service Request*: the client includes the credential in its web service requests.

The header of web service requests in the policy-issuing architecture is smaller than in the capability-issuing architecture as it does not need to include the permission assertions of a capability. However, the complexity of the resource provider is greater, as it needs to maintain a trust relationship with the policy service and keep policies up to date.

2.2 Business Process and Distributed Workflow Security

There are two streams of research and development in the area of security for business processes and distributed workflows: (1) integrating access controls in workflow execution and (2) securing execution and control flow integrity. Our solution considers both of these areas of work.

2.2.1 Integrating access controls in workflow execution

The Task-based Authorization Control (TBAC) framework from Thomas and Sandhu [23] is an extension of the Role-based Access Control (RBAC) concepts [22]. TBAC authorizations are granted and revoked based on when tasks are scheduled and performed. Capabilities are therefore valid only for the duration of a task. TBAC's authorization policy extensions are two fields for activating or deactivating authorizations at runtime. A TBAC authorization therefore has the form $\langle s, o, a, \text{usage } u, \text{authorization-step } as \rangle$. This states that an authorization such as $\langle s, o, a \rangle$ is only valid when contained in the current authorization-step as . Secondly, authorizations are conditioned by a usage or validity count u , specifying the number of times the authorization can be granted in the workflow.

Kang, Park and Froscher [12] consider a similar solution for inter-organizational workflows. The execution environment assumed in their work is CORBA and IIOP, but the principles of CORBA are comparable to those of web services. Inter-organizational workflows are split into autonomous workflow descriptions and distributed amongst the various organizations. Using a capability-issuing approach to security (recall Figure 1-(A)), a two step process of assigning capabilities is followed.

2.2.2 Enforcement of execution integrity

Knorr [15] uses Petri nets as the basis for modeling workflows. The activities of a workflow correspond to the transitions in a Petri net, while the workflow state, data-stores and control-flow are represented by the places and markings. Mendling et al. [16] approach the challenge by extending the BPEL (Business Process Execution Language) with mappings to RBAC [22].

Montagut and Molva [17] demonstrate how to enforce the execution integrity of distributed workflows using onion encryption techniques. Any partner should only be capable of accessing, reading or modifying the data required to execute task at the time the task is scheduled - i.e. a participant only knows a private key at the point in time when a task is assigned to be executed. Using the multi-layered onion encryption technique, a workflow is mapped to

an onion structure and the layered encryption scheme depends on the type of transition between tasks i.e. SEQUENCE, AND-SPLIT, AND-JOIN, OR-SPLIT and OR-JOIN. Biskup [3] et al. present a similar approach towards securing execution orders of composite web services. They use the concept of a container that is passed along the execution path of the workflow as opposed to layered encryption.

3 System Architecture

This section present a system architecture for VO management [5], motivated by the requirements presented in the introduction of the paper. Moreover, the security-aspects of VO management are highlighted, building on the related work presented in Section 2. Subsection 3.1 first describes the components in the architecture and subsection 3.2 describes the interaction messages and data structures that support the VO management protocols.

3.1 Components

Figure 2 provides an overview of the components in the architecture, forming a VO management infrastructure.

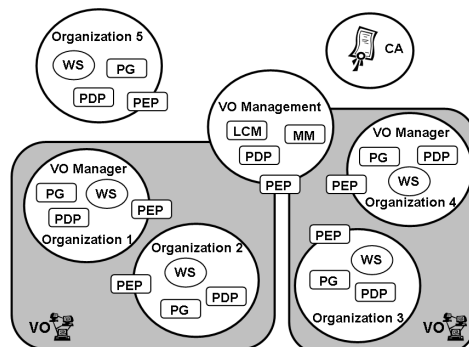


Fig. 2. System Architecture

The components depicted in Figure 2 are described below:

- **Web Service (WS):** The granularity of resources provided in the infrastructure is at the level of web services. Web services enable each participant organization to provide access to their internal functionality and data to other participants in a VO.
- **Policy Generator (PG):** The PG generates authorization policies using the method described in Section 4.3. Its inputs, processing and outputs are

therefore described in section 3.2, step (4).

- **Policy Enforcement Point (PEP)**: We implemented a message-level PEP that not only intercepts and protects the web services that organizations provide to the VO, but also the VO management web services (see section 5.3). Authentication is based on validation of X.509v3 certificates signed by a trusted certificate authority (CA) in the infrastructure or by validating SAML [4] assertions.
- **Policy Decision Point (PDP)**: We have also implemented an extension to a standard PDP in order to make decisions based on control-flow aware security policies as described in Section 4. The policy decision point (PDP) receives the caller's identity, attributes and the web service call details from the PEP. It then evaluates the stored policies and returns either a grant or deny decision to the PEP.
- **Lifecycle Manager (LCM)**: This service is part of the VO management and it allows the creation and deletion of VOs. It also stores the choreography of the VO. The lifecycle manager issues attribute credentials to the creator of a VO.
- **Membership Manager (MM)**: The membership manager, as part of the VO Management, assigns organizations to business roles. For this it also issues tokens to the requestor which can be forwarded to the appropriate member. The security significance of the VO management services are detailed in Section 5.

Before joining a VO, an organization must obtain a global identity certificate. The CA issues this certificate to the organization signed with the private key of the trusted root certificate in a public-key infrastructure (PKI). Thereby the organization becomes part of the VO Infrastructure and is allowed to create VOs. This assumption is also made about every potential participant in a VO, before they can participate in VO-related interactions.

3.2 Interactions and Data Structures

This section describes the interactions and data structures used to implement the VO management and security protocols.

- (1) The process of creating a VO starts with one organization (a VO manager) designing a business process choreography that defines the interactions between participants in the VO. This is done before knowing the identities of participants. We used the Web Services Choreography Description Language (WS-CDL) [2,13] here, as it is an emerging, XML-based standard for describing message-based interactions between web services. Further concepts for using choreography for inter-organizational workflows is described by van der Aalst and Weske [24].

- (2) The VO manager performs a search for participants based on the roles specified in the choreography. The membership manager **MM** and lifecycle manager **LCM** support the VO manager in this process. We do not describe these details here but assume that for each role in the choreography that the VO manager now has a corresponding end point reference of a membership service exposed by each participant.
- (3) For each participant, the VO manager then distributes the relevant parts of the choreography, similar to the partitioning of a public workflow described by van der Aalst and Weske [24]. Along with the partitioned choreography, the relevant service descriptions and public key certificates of participants that need to interact (including the VO manager's) are distributed. Service descriptions are Web Service Description Language (WSDL) documents, while certificates are X.509v3 [9].
- (4) As shown in Figure 2 each participant resource provider has a policy generator **PG** installed. On receiving the choreography, WSDLs and X.509 certificates from a trusted VO manager, the **PG** executes a policy-generating algorithm and installs the access controls on the local PDP. The public key certificates of each participant with whom it needs to interact are then installed at the PEP, as these are required for encrypting outgoing calls. Note that as policies are locally generated, they can be in any format; we used XACML as a convenience, and for purposes of future, internal portability and interoperability of policies.

With the authorization policies and certificates installed at the PDP and PEP respectively, the VO is prepared to execute the choreography. The control-flow aware authorization policies ensure that the execution integrity is maintained using a simple policy state model, as described in Section 4.

4 Control-Flow Aware Authorization Policy Derivation

Our policy derivation algorithm [20] uses the control flow of the choreography in order to minimize the time a policy is enabled. In addition to extracting only the relevant `<InterAction>` elements and enabling them over the lifetime of the choreography, we also use the control-flow of the choreography to trigger the enabling and disabling of the policies. Thereby we achieve a closer approximation of the least privilege principle than the straight-forward interpretation.

4.1 Advantage of Control-Flow Aware Authorization

During the course of enacting a business process many interactions take place and many web services are called on each site. Authorizations at the business process level enable a VO member to access all of these services during the entire life-time of the business process. Even if the service has been accessed and does not need to be accessed again, it is still possible for the VO member to access it. The set of allowed services at any given point in time of the business process is static and maximal.

With control-flow aware authorizations, service access is forbidden after the services have been accessed and are not needed anymore. Services are enabled at the closest possible point in time before their use, such they can only be accessed during that time. Policies which control service access are enabled only for a limited amount of time, i.e. we extend the least privilege principle to time. The set of allowed services (i.e. active policies) at any given point in time of the business process is dynamic and minimal.

4.2 Extended Authorization Policies

We extended the specification of authorization policies in order to develop a mechanism for supporting the control-flow of a choreography. Instead of representing the run-time control-flow (and monitoring it with a second component) we have chosen to represent the static control-flow and extend the policy enforcement and decision with two mechanisms to track the control-flow. Each authorization policy is labelled with two additional fields: one set of policies that are enabled and one with policies that are disabled after the policy has been successfully matched. Let $l_{enable} = \{policy - id_1, policy - id_2, \dots, policy - id_n\}$ be the set of policies to enable and $l_{disable} = \{policy - id_1, policy - id_2, \dots, policy - id_m\}$ be the set of policies to disable. Additionally we store the *policy-id* of each policy as a unique identifying integer and the state of the policy. The state of a policy can be either *enabled* or *disabled*. Disabled policies are not considered when making policy decision, but they have already been created and can be activated on-demand. This allows us to create all policies initially and then reference them by policy-id for enabling and disabling. The life-cycle of a policy is depicted in figure 3. Furthermore, a policy might be enabled and disabled multiple times during the execution of a choreography. Summarizing a policy is a 7-tuple of $\langle policy - id, s, o, a, l_{enable}, l_{disable}, state \rangle$ where *s*, *o* and *a* are the usual authorization policy elements mentioned above.

The sets l_{enable} and $l_{disable}$ are evaluated by the PDP after a policy has been successfully matched, i.e. the PDP evaluates all enabled policies in order,

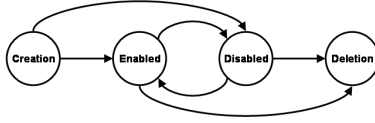


Fig. 3. Policy State Model

comparing them to the request, and, on the first policy matching subject s , object o and action a it allows access. All our policies are *grant* policies specifying an allowed access. The PDP has an implicit *deny* policy that is used for all unmatched requests. Since we only have *grant* policies there is no conflict specified by our policies. Furthermore, if all policies are generated by the policy generator they are non-overlapping. After a policy has been successfully matched to the request the PDP processes the set l_{enable} of this policy and enables all policies in the set, followed by the set $l_{disable}$ of this policy disabling all policies in the set. This implies that a policy might disable itself after successful matching. Our algorithm ensures that no policy appears in both sets (l_{enable} and $l_{disable}$), such that the order of evaluation causes no conflict. Using the Extensible Access Control Markup Language (XACML) as the policy definition language in our implementation, we simply include a new condition and `is-enabled` function in each policy encoded as a `<Condition FunctionId="urn:...:function:is-enabled">` tag. Function `is-enabled` is a simple boolean evaluation function of the PDP that interrogates the l_{enable} and $l_{disable}$ sets.

Whether the enabling and disabling of policies needs to interrupt the evaluation of access requests depends on the implementation of the PDP. Our PDP implementation sequentially scans the policies and we can simply add and remove policies within a synchronized thread.

4.3 Automatic Policy Generation

In this section we will describe the algorithm to populate the enable and disable sets of each policy for each control-flow construct. WS-CDL offers four activities for control-flow: `<Sequence>`, `<Choice>`, `<Parallel>` and `<WorkUnit>`. We will describe the algorithm for populating the enable and disable set with the most basic examples. The extensions to more complex control-flows follow by simple construction.

`<Sequence>` places two or more activities in sequence. Let a, b be two activities, then `<Sequence>a, b</Sequence>` states that a must occur before b . For simplicity we skip the hierarchical nature of WS-CDL and assume wlog that a and b are interactions. Let pol_a be the policy id for an interaction a and pol_b be the policy id for an interaction b . Recall that a policy id uniquely identifies a policy.

If a and b are in sequence, i.e. there is an activity $\langle \text{Sequence} \rangle a, b \langle / \text{Sequence} \rangle$, then the l_{enable} and $l_{disable}$ sets of the policies of ids pol_a and pol_b are as follows:

	pol_a	pol_b
l_{enable}	pol_b	
$l_{disable}$	pol_a	pol_b

$\langle \text{Choice} \rangle$ states that the next activity is one of a choice of multiple. $\langle \text{Choice} \rangle a, b \langle / \text{Choice} \rangle$ states that the next activity is either a (exclusively) or b . Let c be the predecessor of an example choice element $\langle \text{Choice} \rangle a, b \langle / \text{Choice} \rangle$ and d be its successor. Let pol_a, pol_b, pol_c and pol_d be the corresponding policy ids. The enable and disable sets for the policies are then as follows:

	pol_a	pol_b	pol_c	pol_d
l_{enable}	pol_d	pol_d	pol_a, pol_b	
$l_{disable}$	pol_a, pol_b	pol_a, pol_b	pol_c	pol_d

$\langle \text{WorkUnit} \rangle$ allows to repeat the block of contained activities 0 or more times. $\langle \text{WorkUnit} \rangle a \langle / \text{WorkUnit} \rangle$ states that activity a is executed 0 or more times. Let c be the predecessor of the above example with interactions only and d be its successor. The enable and disable sets for the policies are then as follows:

	pol_a	pol_c	pol_d
l_{enable}		pol_a, pol_d	
$l_{disable}$		pol_c	pol_a, pol_d

In case a is a sequence $a_1 a_2$, then a_1 enables a_2 and disables d while a_2 enables a_1 and d . This cancels each other out in case of one activity a only.

The activity $\langle \text{Parallel} \rangle$ is different from other activities, since it introduces parallelism. $\langle \text{Parallel} \rangle a, b \langle / \text{Parallel} \rangle$ states that activities a and b are to be executed in parallel. The workflow is re-synchronized after the parallel activity, i.e. before it proceeds to the next activity both a and b have to be finished.

Since the sequence of completion of a and b is unknown, it is difficult to express the enable and disable sets without state. We therefore encode the state in additional policies. Let there be n parallel interactions in a parallel activity element and for simplicity number them from a_0 to a_{n-1} . We then have a policy for each not yet completed interaction for each set of completed interactions. Here is an example: At the beginning no interactions has been completed,

therefore there are n policy for the set $\{\}$. Next any interaction may completed, say a_i . Then there are $n - 1$ policies (for activities $a_0, \dots, a_{i-1}, a_{i+1}, \dots, a_{n-1}$) for the set $\{a_i\}$. And so forth.

We can represent the set of completed activities as an n -bit bit-mask. Reading this as a binary representation of a number, each completed set is assigned a number between 0 and $2^n - 1$. Let $pol_{x,i}$ be the policy id for policy with completed set x and interaction a_i . Policy $pol_{x,i}$ then contains policies $pol_{x+2^i,j}$ for $j \in [0, n - 1]$ in its enable set and policies $pol_{x,i}$ for $i \in [0, n - 1]$ in its disable set. Note that not all indices i, j are available for any completed set x . In other words a policy match for interaction a_i advances the set of completed activities from x to $x + 2^i$.

Let c be the predecessor of the parallel activity and d its successor. Activity c contains policies $pol_{0,i}$ for all $i \in [0, n - 1]$ in its enable set. No policies for completed set $x = 2^n - 1$ are generated. Instead, each policy $pol_{2^n-1-2^i,i}$ for any $i \in [0, n - 1]$ contains d in its enable set. The disable sets are identical to the intermediate state policies and contain just itself.

The example for a parallel activity $\langle \text{Parallel} \rangle a, b \langle /\text{Parallel} \rangle$ has the following enable and disable sets.

	$pol_{0,a}$	$pol_{2,a}$	$pol_{0,b}$	$pol_{1,b}$	pol_c	pol_d
l_{enable}	$pol_{1,b}$	pol_d	$pol_{2,a}$	pol_d	$pol_{0,a}, pol_{0,b}$	
$l_{disable}$	$pol_{0,a}, pol_{0,b}$	$pol_{2,a}$	$pol_{0,a}, pol_{0,b}$	$pol_{1,b}$	pol_c	pol_d

The disadvantage of encoding the state of completed interactions is exponential explosion of policies. If there are n parallel interactions, then the number of policies is $\sum_{i=0}^{n-1} i \binom{n}{i} = n2^{n-1}$.

4.4 Local View

Another design choice of our control-flow enforcement in choreographies is that we work without global state. Not all participants have the same view on the state of the choreography. Parties that are not currently involved in the interactions only approximate the current view. This frees us from the necessity of exchanging additional messages between all participants after each step of the choreography. The only synchronization overhead is given by the choreography itself.

We are aware that this relaxes the achievement of the least privilege principle, but the choice was made in order to keep the distributed system manageable.

Global state is very difficult to maintain and introduces many extra steps and delays into the protocol. Instead we chose to have each party compute its local view from the choreography itself.

This proceeds as follows: Imagine the example choreography from Figure 4. We now compute the local view of party *A* Alice.

First, Alice filters all interactions that do not have herself as a destination, i.e. only interactions remain that are intercepted by her PDP. An improvement in the precision of our algorithm can be made by introducing outgoing filtering with the same PDP. Then all interactions that have herself as a source or destination remain unfiltered.

Second, Alice removes all unnecessary nodes and control flow elements. Note that in the example choreography Alice removes the `<Parallel>` control flow element, since she is only involved in one branch. Therefore she does not notice the parallel interactions and can act as if it were single threaded. Alice's view on the example choreography is depicted in Figure 5.

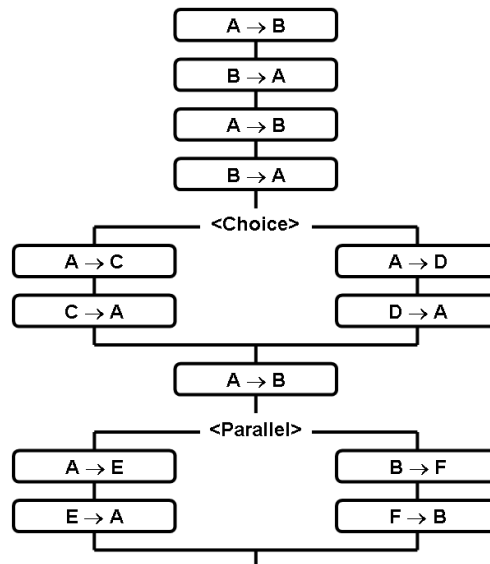


Fig. 4. Example Choreography

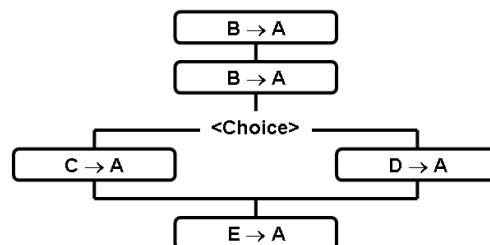


Fig. 5. Alice's View of the Choreography

Third, Alice computes her control-flow aware policies on her local view of the choreography as described in Section 4.3. Note that Alice e.g. does not wait for her calling the first web service at B Bob in the top most interaction of the example choreography. Instead she enables the policy allowing Bob to call her. This is the mentioned breach of the least privilege principle, since Bob is given a bigger time window than would be necessary with global (complete) state. We tolerate this breach for the gain of a significant simpler enactment of the choreography.

5 VO Management Security

The other important novel feature of our VO security architecture is the protection of the VO management services. While usually VO management services are protected by their own security mechanisms which are directly implemented in the code of the VO management services, we chose to implement security in the same external mechanism that protects the web services. VO management security is crucial, since its compromise endangers the security of the services offered within each VO. Therefore security for VOs can only exist in a comprehensive architecture that spans VO management services and services offered within a VO.

The choice to implement VO management security with the same PEP/PDP mechanism as the VO services improves over previous research in three aspects. First, the resulting system is easier to administer, since the administrator only needs to master one policy language and there is only one policy administration interface. Second, VO management becomes flexibly manageable via policies as opposed to hard-coded security in the VO management services. And third and foremost, the trusted computing base, i.e. the security critical part of the VO system, decreases in code size. Audits and implementation effort should decrease in consequence as well as making the secure enactment of VOs more economic.

5.1 VO Management Interface

Although the management interface of the VO infrastructure services supports the entire lifecycle of the VO we chose to focus on the following security critical operations for brevity:

- **Lifecycle Manager (LCM):**
 - **createVO:** creates the VO identifier used by all members
 - **deleteVO:** invalidates the VO identifier

- **getChoreography**: returns the business process choreography
- **Membership Manager (MM)**:
 - **getRoles**: returns a listing of all valid roles and their assigned members
 - **assignRole**: assigns a role to a member in the VO to be performed in the choreography
 - **removeRole**: removes a role in the business process that has been previously assigned to an identified member

5.2 VO Management Policies

Subject	Object	Action
*	LCM service	createVO()
VOMANAGER	LCM service	deleteVO()
BP-ROLE	LCM service	getChoreography()
VOMANAGER	MM service	assignRole()
VOMANAGER	MM service	removeRole()
BP-ROLE	MM service	getRoles()

Fig. 6. Policies for VO Management Security

As a starting point we present the set of policies from Figure 6 for VO management security. They implement the following, rather simple security model: Each VO has a VO manager that is responsible for all administration tasks who can access all administration services (MM and LCM) and perform the necessary operations. VO members may query the management services (MM) to get information about the VO they are part of, i.e. the list of the roles, etc. On the other hand, non-members should be prohibited from obtaining any information about the VO.

We give an example how these policies should be interpreted: A VO member tries to retrieve the choreography (e.g. for generating the necessary policies) from the LCM and calls the `getChoreography` method with the VO-id as parameter. He presents a credential [4] with this VO-id for a role in the choreography, any of which matches the placeholder *BP-ROLE*, and consequently access is granted. He should be denied access, if he cannot present such a credential and thereby prove that he is a member of the VO. Further details of this verification procedure and the extraction process can be found in the next section.

5.3 PEP Implementation

A preliminary version of the PEP implementation has been presented in [14]. The PEP implementation adds confidentiality of roles and VO-ids and ties the parameters of a (VO management) web service call to the roles of the credentials. The goals of the PEP implementation besides providing confidentiality, integrity and authenticity to web service calls are to provide security for VO management, i.e. enforce the VO management security policies, and to minimize information leakage to outsiders of a VO. The novel means to achieve these goals are two-fold: First, the PEP encrypts and later decrypts all confidential information in tokens and second, it extracts the VO-id from the web service call and matches it to the credentials. We claim that this achieves both goals mentioned and therefore provides the basis for a secure and confidential VO system including VO management. The PEP in combination with an essentially standard PDP is our single security solution for all aspects of VO security in our architecture.

Input: web service call in SOAP XML from Alice A to Bob B ; a set of pairs of keys i, k_i ; a set of credentials $c_j : name, i, role, signer, signature$; a certificate for Alice $C_A : name, publickey, signature$; a private key for Alice K_A ; a (verified) certificate for Bob $C_B : name, publickey, signature$.

```
for each credential  $c_j$ 
  encrypt  $c_j.i$  with  $k_i$ 
  encrypt  $c_j.role$  with  $k_i$ 
  add  $c_j$  to call (header)
add certificate  $C_A$  to call (header)
sign call (body) with  $K_A$ 
encrypt call (body) with  $C_B.publickey$ 
```

Fig. 7. Algorithm for outgoing web service calls

Each party has a set of keys k_i ; one for each VO i it is part of. These keys k_i are symmetric keys and revocation of these keys from members leaving the VO is not considered in our architecture, i.e. they are never revoked. These keys intend to protect the VO information against outsiders, e.g. man-in-the-middle, and not VO members or former members. Former members are aware which role a partner is playing in the VO and could divulge that information whether it is encrypted or not.

Our PEP has been implemented as a handler which automatically intercepts all web service calls. Such a handler can be installed via configuration files such that web service do not need any modification at all when incorporated into our VO security architecture. This has been a major design goal, since services are often designed for intra-enterprise use and should not need to be retrofitted in order to join VOs. The absence of any modification of the

service makes web service calls VO oblivious which requires some fundamental changes to the PEP architecture. E.g. it is not possible to actively select (or negotiate) the credentials to be put into a web service call.

Input: web service call in SOAP XML from Alice A to Bob B ; a set of pairs of keys i, k_i ; a private key for Bob K_B ; a (trusted) root certificate $C_{root} : name, publickey$.

```

decrypt call (body) with  $K_B$ 
extract certificate  $C_A$ 
verify certificate  $C_A$  with  $C_{root}$ 
verify signature of call with  $C_A.publickey$ 
o := web service
a := method
parse call (body) for special XML-wrapped parameter VO-id
if  $\exists$  parameter VO-id
    id := parameter VO-id
else
    id := null
for each credential  $c_j$ 
    for each key  $k_i$ 
        decrypt  $c_j.i$  with  $k_i$ 
        if  $c_j.i == i$ 
            decrypt  $c_j.role$  with  $k_i$ 
            verify  $c_j$  with  $C_{root}$ 
            if id == null or id == i
                s :=  $c_j.role, i$ 
                if PDP(s, o, a) == accept
                    forward call
block call

```

Fig. 8. Algorithm for incoming web service calls

The PEP follows the algorithm in Figure 7 for outgoing web service calls. Its main feature is that it adds all credentials to the web service call, since it cannot distinguish to which VO the call belongs, but protects their confidentiality by encrypting them. The call will be intercepted at the receiving site by the same PEP which follows the algorithm in Figure 8 for incoming web service calls. Besides the standard decryption and signature verification it extracts the VO-id parameter where care needs to be taken that the same parameter is extracted to regular parsing such that XML rewriting attacks are prevented. Then the information of each valid credential is sent to the PDP and if it accepts the call, then the call is forwarded to the resource web service while all other calls are blocked. There is also a modification in the use of the PDP compared to other architectures: Only accept calls are forwarded and an implicit deny policy is implemented by the PDP, because there might be false rejects due to additional VOs the partners are in that do not belong to this specific call.

The verification procedures of credentials and certificates have been described in simplified form compared to the actual implementation in order to clarify the description. The implementation allows chaining of certificates as well as credentials and the chains are followed during verification.

5.4 Implementation using XACML

In order to implement, integrate and evaluate the concept in a larger VO management system, we have modified and integrated with the standard XACML[18] schema and architecture. The relation of our concept and solution to the XACML standard is shown in Figure 9. The PIP (Policy Information Point) was treated as a module of the MM and LCM components in the broader architecture.

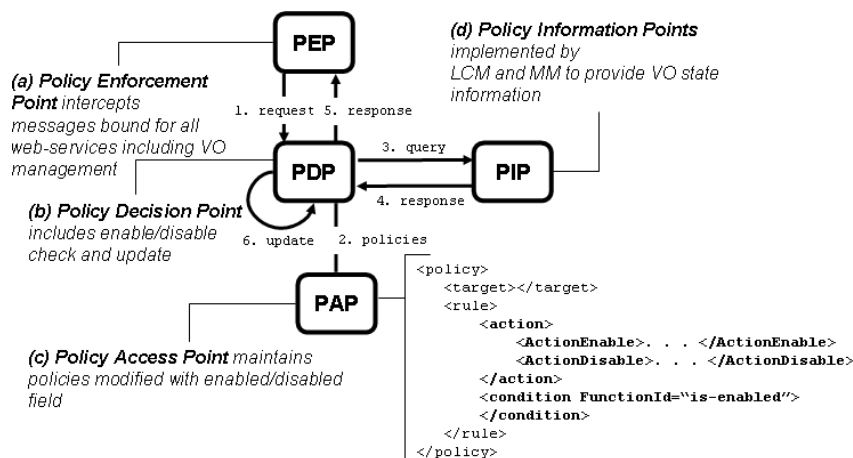


Fig. 9. Implementation of solution using XACML architecture

There were two methods for implementing using XACML. Method 1 had the goal of not modifying the XACML schema. In this method, the PDP was responsible for maintaining a separate set of policies other than the pure-XACML policies maintained by the PAP. The advantage of this methodology is the ability to conform with the XACML standard and maintain interoperability with other PDPs. The drawback was the additional logic and policy-state maintenance now extending the functionality of the PDP. Method 2, shown in Figure 9 sought to compensate for this; in method 2 we updated the XACML schema to include an `is-enabled` function as a condition within each policy, as well as two actions for enabling `ActionEnable` and disabling `ActionDisable` sets of policies. While we could maintain the standard functionality of the PDP, the XACML functions and actions no longer conformed to the standard.

6 Conclusions

We showed our security architecture for Virtual Organizations of Business Web Services which improves over the state of the art with its control-flow aware policy generation and its single security mechanism for VO management and web services.

We automate the generation of access controls with our policy generator. It adheres to the least privilege principle even in time in the sense that it enables only a close to minimal set of policies necessary at a given point in time of the choreography. We maintain the autonomy of each participant by locally executing the policy generator solely based on an agreed choreography. Every participant is in charge of the policies on his own policy decision point (PDP). We achieve control-flow awareness of policies in the PDP without global state, i.e. we do not introduce additional calls for updates after message exchanges. Instead each participant maintains a local view on the control-flow. Besides the policy generator only the policy enforcement point (PEP) and the PDP are part of the trusted computing base. We outsource the security enforcement of VO management services to the enhanced PEP/PDP combination and thereby reducing the size of the trusted computing base and making VO management security more flexible and easier to administer. This PEP/PDP combination addresses the needs of all critical components of a VO security architecture in a single security solution. It not only focuses on the protection of the resources (web services), but extends to VO management and thereby ensures that the entire infrastructure is protected.

We intend to extend the security model of our architecture to include delegation and enhanced VO management services, but without sacrificing its stringent security properties.

7 Acknowledgements

This work strongly benefited from our collaboration with Hauke-Hendrik Vagts, Rafael Deitos and Andreas Schaad.

References

- [1] R. Alfieri, R. Cecchini, V. Ciaschini, L. dellAgnello, A. Frohner, A. Gianoli, K. Lorentey, and F. Spataro. VOMS: An Authorization System for Virtual Organizations. *Proceedings of the 1st European Across Grids Conference*, 2003.

- [2] A. Barros, M. Dumas, and P. Oaks. A Critical Overview of the Web Services Choreography Description Language. *BPTrends*, 2005.
- [3] J. Biskup, B. Carminati, E. Ferrari, F. Müller and S. Wortmann. Towards Secure Execution Orders for CompositeWeb Services. *Proceedings of International Conference on Web Services*, 2007.
- [4] S. Cantor, J. Kemp, R. Philpott, and E. Maler. Security Assertion Markup Language (SAML) 2.0 Specification. *OASIS*. Available at <http://docs.oasis-open.org/security/saml/v2.0/saml-2.0-os.zip>, 2005.
- [5] R. Deitos, F. Kerschbaum, P. Robinson, and J. Haller. A Comprehensive Security Architecture for Dynamic, Web Service Based Virtual Organizations for Businesses . *Proceedings of the ACM Secure Web Services Workshop*, 2006.
- [6] Y. Demchenko, L. Commans, C. de Laat, M. Steenbakkens, V. Ciashini, and V. Venturi. VO-based Dynamic Security Associations in Collaborative Grid Environment. *Proceedings of Workshop on Collaboration and Security of the International Symposium on Collaborative Technologies and Systems*, 2006.
- [7] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Int. Journal of High Performance Computing*, 2001.
- [8] M. Harrison, W. Ruzzo, and J. Ullman. Protection in Operating Systems. *Communications of the ACM 19(8)*, 1976.
- [9] R. Housley, W. Polk, W. Ford, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. *IETF RFC 3280*. Available at <http://www.ietf.org/rfc/rfc3280.txt>, 2002.
- [10] W. Johnston, S. Mudumbai and M. Thompson. Authorization and Attribute Certificates for Widely Distributed Access Control. *Proceedings 7th IEEE International Workshop on Enabling Technologies*, 1998.
- [11] M. Juric, I. Rozman, B. Brumen, M. Colnaric, and M. Hericko. Comparison of Performance of Web Services, WS-Security, RMI, and RMI-SSL. *Journal of Systems and Software 79(5)*, 2006.
- [12] M. Kang, J. Park, and J. Froscher. Access control mechanisms for inter-organizational workflow. *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies*, 2001.
- [13] N. Kavantzias, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto. Web Services Choreography Description Language Version 1.0. *W3C*. Available at <http://www.w3.org/TR/ws-cdl-10/>, 2005.
- [14] F. Kerschbaum, R. Deitos, and P. Robinson. Securing VO Management. *Proceedings of the 4th International Conference on Trust, Privacy & Security in Digital Business*, 2007
- [15] K. Knorr. Dynamic access control through Petri net workflows. *Proceedings of the 16th Annual Computer Security Applications Conference*, 2000.

- [16] J. Mendling, M. Strembeck, G. Stermsek, and G. Neumann. An Approach to Extract RBAC Models from BPEL4WS Processes. *Proceedings of the 13th IEEE International Workshops on Enabling Technologies*, 2004.
- [17] F. Montagut, and R. Molva. Enforcing Integrity of Execution in Distributed Workflow Management Systems. *Proceedings of IEEE International Conference on Services Computing*, 2007.
- [18] T. Moses. eXtensible Access Control Markup Language 2 (XACML) Version 2.0 Specification. OASIS. Available at http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf, 2005.
- [19] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke, A Community Authorization Service for Group Collaboration. *Proceedings of the 3rd IEEE International Workshop on Policies for Distributed Systems and Networks*, 2002.
- [20] P. Robinson, F. Kerschbaum, and A. Schaad. From Business Process Choreography to Authorization Policies. *Proceedings of the 20th IFIP WG 11.3 Working Conference on Data and Applications Security*, 2006.
- [21] P. Samarati and S. De Capitani di Vimercati. Access Control: Policies, Models, and Mechanisms. *Foundations of Security Analysis and Design. LNCS 2171*,, 2001.
- [22] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role Based Access Control Models. *IEEE Computer 29(2)*, 1996.
- [23] R. Thomas, and R. Sandhu. Task-Based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-Oriented Authorization Management. *Proceedings of the 11th IFIP International Conference on Database Security*, 1998.
- [24] W. van der Aalst, and M. Weske. The P2P Approach to Interorganizational Workflows. *Proceedings of the 13th International Conference on Advanced Information Systems Engineering*, 2001.

A Complex Example

This section gives an example of a more complex choreography and its derived control-flow aware authorization policies. Consider the following example with its enable and disable lists below.

```

<WorkUnit>
  <Sequence>
    a,
    <WorkUnit>
      <Choice>
        <Sequence>b, c</Sequence>,

```

```

    d
  </Choice>
</WorkUnit>,
  e
</Sequence>
</WorkUnit>

```

	pol_a	pol_b	pol_c	pol_d	pol_e
l_{enable}	pol_b, pol_d	pol_c	pol_b, pol_d, pol_e	pol_e	pol_a
$l_{disable}$	pol_a	pol_b, pol_d, pol_e	pol_c		pol_b, pol_d, pol_e