IDIS (2009) 2:137–154
DOI 10.1007/s12394-009-0031-5

Delegation of access rights in multi-domain service compositions

Laurent Bussard · Anna Nano · Ulrich Pinsdorf

Received: 12 January 2009 / Accepted: 29 September 2009 / Published online: 1 December 2009

Abstract Today, it becomes more and more common to combine services from different providers into one application. Service composition is however difficult and cumbersome when there is no common trust anchor. Hence, delegation of access rights across trust domains will become essential in service composition scenarios. This article specifies abstract delegation, discusses theoretical aspects of the concept, and provides technical details of a validation implementation supporting a variety of access controls and associated delegation mechanisms. Abstract delegation allows to harmonize the management of heterogeneous access control mechanisms and to offer a unified user experience. The authors observe standardization efforts to reduce application and domain-specific delegation mechanisms, but this variety is very unlikely to completely disappear.

Keywords Access control · Authorization · Composite services · Delegation · Identity · Service oriented architectures · SOA

Why is delegation so cumbersome today?

Developing new services and applications typically involves composition of existing services. Composition can result in a mash-up service, a workflow, or a plain application. Typically the services to be composed come from different service

L. Bussard · A. Nano · U. Pinsdorf (✉)
European Microsoft Innovation Center, Ritterstr. 23, 52072 Aachen, Germany
e-mail: ulrich.pinsdorf@microsoft.com

L. Bussard
e-mail: lbussard@microsoft.com

A. Nano
e-mail: annaw@microsoft.com

providers. Such an application could end up with combining services e.g. from Amazon, Microsoft, and Google. Each of these service providers controls its own trust domain and in general there is no mutual trust between service providers. We refer to this as a *multi-domain scenario*. Figure 1 gives an example use case for delegation in a multi-domain service composition.

Usually the problem of mutual trust establishment is addressed by means of manual exchange and identity tokens, such as X.509 certificates. The drawback is that this gives the composition a somewhat static attitude. Service compositions have to find a trade-off between level of security, dynamic composition, and manageability. However, when it comes to personal or even business-critical information that is shared, appropriate security mechanisms are essential. Since most scenarios do have security requirements, it becomes difficult to compose services from different trust domains because a) access control (AC) models are heterogeneous and b) there is no common trust anchor. These are major roadblocks for service composition.

Further, we believe that due to the steadily growing number of individual online services and their easy composability, delegation in multi-domain service composition will become an important requirement.

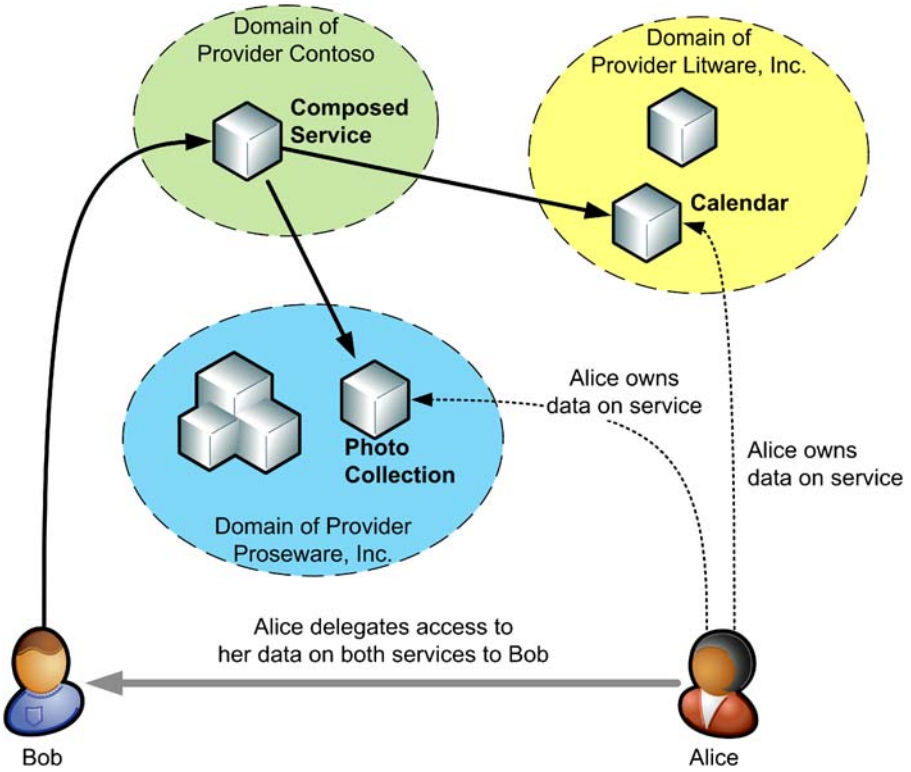

Fig. 1 Example use case for delegation in multi-domain service composition. Bob wants to invoke a composed service that aggregates Alice's data stored on two other services. All services are offered by different service providers and may have different access control mechanisms in place. We assume that there is no common trust anchor between all three providers. How shall Alice delegate access rights to her data to Bob in an easy, user-friendly and secure way?

Delegation of access rights generally requires user's approval. This leads us to the essential aspect of the user experience for delegation in composite services. There is a large diversity of access control mechanisms, e.g. X.509-based access control lists, SecPAL (Becker et al. 2007), username/password lists, and XACML (Moses 2005). All these mechanisms require different handling, we call them *management actions*, to be performed by the user. Additionally more and more industry APIs for authentication are available, e.g. Amazon Web Services Authentication, Facebook Authentication, Flickr Authentication API, Microsoft Live ID Delegation, and OpenID. All authorization mechanisms have their individual strengths and may be applied to guard individual services. Hence, the chance that a user deals only with one access control mechanism is very unlikely. In reality the user is confronted with a number of these access control mechanisms. This heterogeneity is a real issue. The typical user will not want to learn how to configure each of them in order to protect all her services, respectively learn how to give access to other persons. We see efforts both in research (Patil et al. 2007) and standardization (Fitzpatrick et al. 2007; Moses 2005; OAuth Core Workgroup 2007) to make different access control mechanism interoperable. This may decrease the diversity, but it is obvious that the variety of access control mechanisms will never totally disappear.

What end-users and software developers need is an abstraction layer that offers delegation and hides the details of different access control mechanisms. This abstraction layer has to be designed in a way that the user has a set of high-level operations at hand which are common to all access control mechanisms. For instance, the user expresses her wish to grant or revoke some right to a specific service and the abstraction maps this to an operation on the access control mechanism of this specific service. This abstraction solves in fact two problems. First the user is able to manage the access control of all her services, which in turn raises the user's confidence and trust in the system. Second, this lays the basis for delegation of access rights in multi-domain service composition scenarios without embedding too many details regarding underlying mechanisms. Hence, it removes the roadblock for multi-domain service compositions. We refer to this abstraction as *abstract delegation*. This article describes and discusses the concepts and implementation of this abstraction layer.

Definition of terms

This section defines the main terms that we will use throughout this article. Figure 1 shows the three actors of our scenario and their relation to each other: the delegator, the delegatee and the resource. Without loss of generality we assume that the delegatee invokes the resource. The resource is managed by the delegator. Moreover we understand all three actors as generalized services; particularly the delegatee and the resource may be part of a service composition. The next paragraph shall define the three terms more precisely.

We define a *resource* as a service that is valuable for a user. Owning a resource does not necessarily mean that user is also hosting the service. Think about an online picture service that stores users' digital picture collections. Although the user does not own the picture service as such, she owns her photo collection at this service.

Other examples are location services, electronic medical records, personal schedule and agenda, etc. In pervasive computing environments, resources are also extended to appliances ranging from lamps e.g. service controlling lighting at user's home, to cars e.g. service unlocking and starting user's car.

Resources generally need mechanisms to enforce access control for obvious security and privacy reasons including confidentiality. Access to those resources can be delegated to third parties e.g. friend, hospital, housekeeping. We define a *delegator* as an authorized entity *a* that can authorize another entity *b* to access a given resource *r* with privileges that were previously explicitly or implicitly agreed between *a* and *b*. We define a *delegatee* as an entity *b* that requests access to a resource *r* and is authorized by delegator *a*. We call *delegation* the act of granting access rights. Delegation could involve the creation of a new credential, uploading a credential of *b* to an access control list, or modifying an existing security policy. We do not focus on a specific mechanism here. Rather we suggest a framework featuring a set of plug-ins where each plug-in allows the configuration of a specific access control mechanism. Besides configuring the access control mechanism of *r* the delegator provides information to the delegatee how to invoke the service. It is important to understand that we do not aim at delegation by means of impersonation, where an entity *b* temporarily acts as entity *a*.

We distinguish between two types of delegation, which should be supported by any delegation mechanism for multi-domain service compositions. A delegation is *delegatee-driven* if the delegatee asks the delegator for access to a resource. The request can be static (defined in metadata of a composite service), dynamic (e.g. required through a call-back mechanism), or upon request (directly required by the delegatee without previous interaction). In contrast, a delegation is *delegator-driven* if the delegator chooses to delegate some rights to a delegatee. Note that the request could come from the delegatee in a non-formatted way that cannot be processed automatically by the delegation system, e.g. as verbal request. The selection of the delegatee can be preselected (e-mail/SMS asking for delegation), based on a list (delegate to a friend/colleague), or based on discovery or enrolment mechanisms. We expect the delegatee-driven delegation to become the most common form in service compositions across multiple trust domains.

Another distinction is the type of delegation mechanisms, i.e. the type of access control mechanism protecting the resource. They can be distinguished in three classes:

- **Chain of Credentials**: The delegator (or delegator-side Security Token Service) creates a new security token that is provided to the delegatee. Examples for this class are SecPAL and SPKI.
- **Policy Manipulation**: The delegator modifies the access control policy of the resource. Examples for this type are modification of authorization policy at a Security Token Service (STS) or at a Policy Decision Point (PDP), or adding an identifier of the delegatee (e.g. X.509 certificate, username, corporate alias, fingerprint) to an access control list (ACL).
- **Hybrid**: Combination of chain of credential and policy manipulation. For instance, the delegator creates a new pair of credentials, registers it at the resource, and provides it to the delegatee. In other cases, the delegator could ask the delegatee to register/create an account before delegation.

This solution covers all three types and most of the mentioned examples.

Abstract delegation

This section presents the architecture and implementation of a general framework that allows delegation. In the opening section we argued, that on one hand users care about privacy and want to control access to their data and services while on the other hand, multiple access control mechanisms and associated management interfaces coexist and lead to incoherent user experiences.

Approach

The abstraction covers delegatee, resource, rights, and revocation. The delegator is thus able to specify — in a composition or using a unified user interface — that an abstract delegatee has some access rights on an abstract resource. This abstract delegation is mapped at runtime to a concrete mechanism. For instance, in order to grant access an XACML policy could be modified or a new SecPAL credential could be created and handed over to the delegatee. Abstract revocation of access rights is also mapped to appropriate mechanisms, e.g. modification of a policy, or adding an entry to a revocation list.

Figure 2 visualizes the conceptual idea of abstract delegation. The delegator defines a delegation relationship between a delegatee and a resource. This is done by means of an appropriate user interface which supports a high-level view and hides the details from the delegator (unless the delegator wants to see them). Thus, the delegator understands what is going on without caring too much about the technical details. For instance, the patient Alice delegates read access for her medical record service to a doctor called Bob (see Fig. 3). The abstraction allows Alice to refer to 'Doctor Bob' instead of to a specific piece of identity information like an X.509 certificate. Similarly, the resource is presented to the delegator in a semantically meaningful way. It is not just an URL, but a rich set of information about the service. The information about a delegatee is collected in a data structure that we call delegatee card; likewise the information about a resource is captured in a resource card.

Resource cards are data containers with all information necessary to access and administrate a service. Resource cards should be issued and signed by resource providers, for instance when a user registers to an on-line medical record. In case of

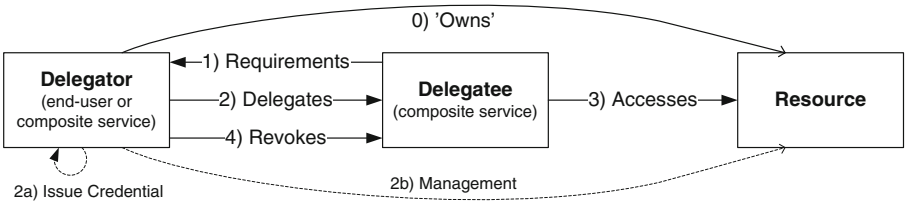

Fig. 2 Generalized scenario with three participants delegator, delegatee, and resource

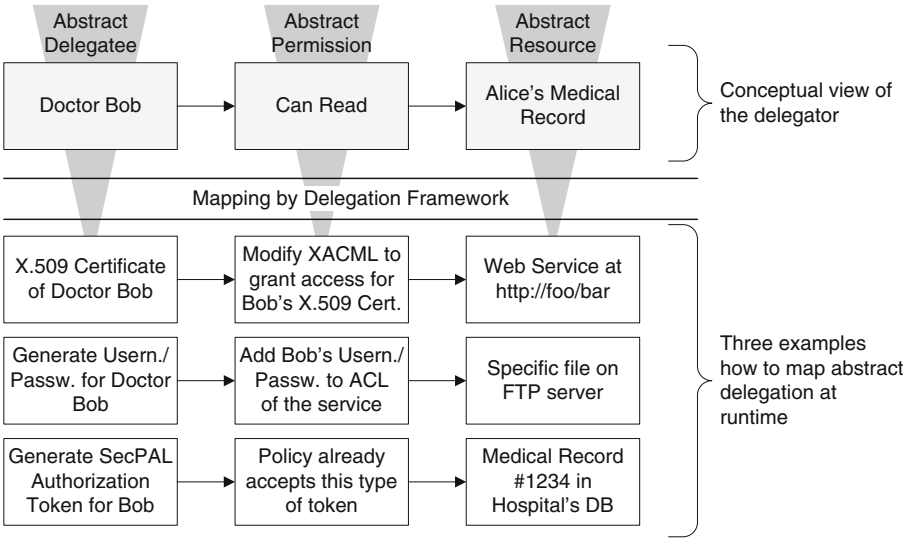

Fig. 3 Mapping between abstract delegation and three concrete actions

legacy services, which obviously do not provide resource cards, we assume that the user or a third party would create the appropriate card. In more detail, resource cards contain a unique resource identifier, visual information (friendly name, icon, description, category of resource), resource description (resource type, resource address, resource credential), and delegation info (max duration, rights that can be delegated, management type, management address, management credential, delegator's credential for management). Resource card attributes can be optional if they are provided as metadata by the resource and management interface. Some data, e.g. user credentials, cannot be delivered by the service provider and hence must be set by the user while installing the card, e.g. a reference to a key store.

Delegatee cards describe the delegatee and contains the following information: a unique subject identifier, visual information (friendly name, icon, category of delegatee), and a list of identifiers (X.509 certificates, user name in a specific scope, fingerprint). It may contain multiple public identifiers for the same party. For instance, Alice may know the X.509 certificate of Bob as well as Bob's fingerprint feature vector; Alice's delegatee card for Bob would contain both identifiers. New identifiers can be created, e.g. in an enrolment process, and existing identifiers can be added, removed or modified. Delegatee cards can either be provided by the delegatee or can be created by the delegator.

The actual mapping from the high-level abstract view to the low-level technical authorization depends on the authorization model of the resource. The resource card contains the information about the service's authorization mechanism. This gives the delegation framework sufficient information to execute what is necessary that the service successfully authorizes the delegatee when he invokes an action on the service. Figure 3 shows three mappings between the high-level delegation and technical realization. Those three mappings are just examples and depend on the technical implementation of the service's authorization system.

Architecture

The architecture that implements the concept of abstract delegation consists mainly of a delegation framework located at delegator's side. Figure 3 shows the involved components and the interactions between delegator, delegatee, and resource. The delegation framework makes the abstraction for the user. It relies on resource cards and delegatee cards.[1] Based on these information it performs all necessary actions to grant access to the delegatee. For the concrete delegation actions it relies on a set of plug-ins. Each plug-in is specialized in a specific authorization mechanism (e.g. SecPAL based policies, XACML, X.509 ACL). The framework needs a plug-in for each authorization mechanism that one of the delegator's resources might utilize. The framework is enhanced with two user interfaces: the *Delegation Selector*, which enables the delegator to select appropriate resource and rights to delegate (see Fig. 5), and the *Delegation Browser*, which offers a holistic view of all ongoing delegations (see Fig. 6).

The delegatee does not need a specific part of the delegation framework, but needs to support the authorization protocol of the resource and has to deal with the credentials provided by the delegator. But this is what the delegatee does even without the delegation framework. The resource and its associated guard do not need to fulfill specific requirements either.

Figure 4 depicts interactions between all actors and components. In step 1, the delegatee requests access to a specific type of resource. Note that the delegator may not have a priori knowledge about the expectation of the delegatee regarding access to resources, e.g. a social network composite service may require access to the user's calendar to search for contacts. In this case, the delegatee (social network) would expose meta-data regarding its expectation on required resources (access to a calendar resource). The *Delegatee Id* as well as the *Resource Description* are provided to the delegation framework. In step 2, the delegation request is compared to available *resource cards*. Resource cards that match the request, i.e. same type of service and supported delegatee, are presented to the delegator that selects the resource he wants to delegate access to (step 3). This step involves the *delegation selector* (see Fig. 5) which combines service discovery and delegation in a CardSpace-like user experience.[2] Step 4 does what is necessary to grant permission to the delegatee to access the resource. The framework calls the plug-in referenced in the resource card, which in turn performs the necessary actions on the resource guard. Depending on the protocol used to manage the resource guard, additional steps may be required for authentication and authorization purpose. The next section will describes existing plug-ins and provide details on a concrete example. The result of the delegation is provided to the delegatee in step 5: the delegator provides the credential to be used for authorization and a description of the resource including its address and the granted permissions. The credential can be a new one (e.g. a SecPAL token) or a reference to a credential of the delegatee (e.g. its own X.509 certificate).

[1] In the pilot implementation the delegation cards and resource cards are stored locally. This has the obvious drawback that they are accessible only on the single machine where they are stored on. The concept could be extended with a cloud based data store for these cards.

[2] See http://www.microsoft.com/net/WindowsCardSpace.aspx

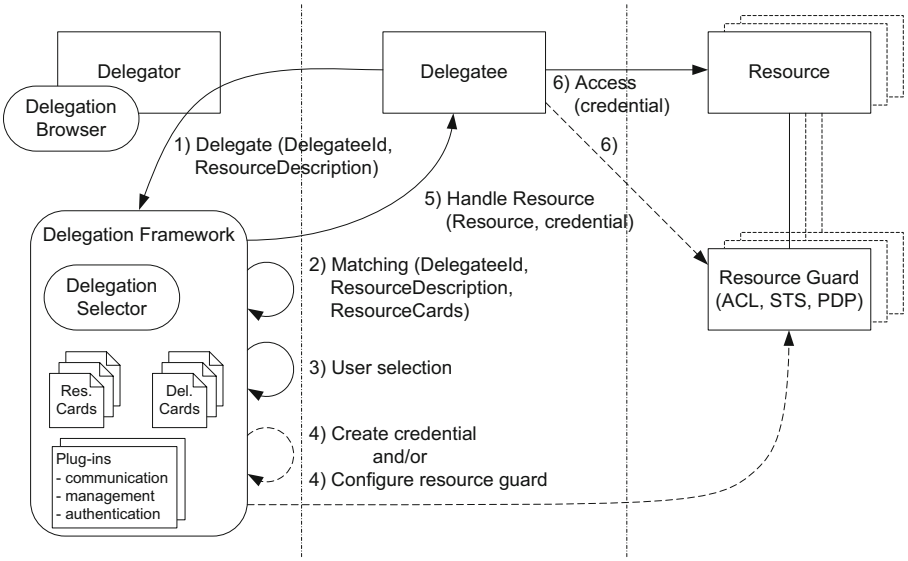

Fig. 4 Components and interactions

Finally, in step 6, the delegatee accesses the resource using the received credential. Additional steps may be necessary depending on the protocol used to access the resource (e.g. WS-Federation, WS-MetadataExchange).

Delegation plug-ins

Since plug-ins map abstract delegation to concrete mechanisms, they have to deal with different aspects of the abstraction:

- *Delegation*: mainly policy manipulation (management operations) vs. chain of credentials (creation of credentials).
- *Management channel*: type of communication (e.g. WS-ResourceTransfer, REST, Plain SOAP) and authentication to management interface.
- *Resource*: type of authentication mechanisms supported by the resource. This is used to ensure that the delegatee gets enough information to access the resource.

The remaining of this section describes the function of plug-ins through an example and next briefly describes few plug-ins we did implement to cover a large variety of delegation mechanisms we are aware of, namely: PKI-based, WS-Trust (STS), username/password-based ACL, Kerberos, claims-based Access Control, and other authentication mechanisms (e.g. biometrics).

Matching in delegation plug-ins

We use the SecPAL plug-in to describe the abstract API and how the mapping from abstraction to a concrete delegation mechanism is done. More information on the API can be found in a technical report (SeCSE 2008).

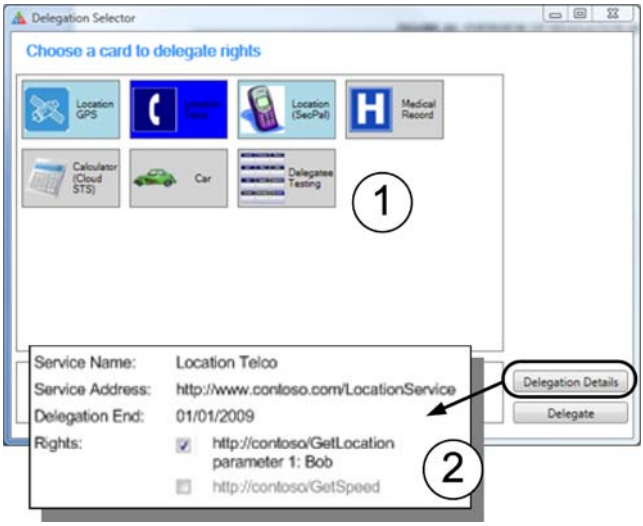

Fig. 5 The CardSpace-like Delegation Selector allows delegating access to own resources

For the sake of simplicity, we assume a scenario with basic resources and policies. The resource is the medical record service[3] (MedService) of the Alice (delegator) implemented as a file server with two directories: *file://Alice/general* and *file://Alice/confidential* containing general health information (e.g. blood group) and more confidential data (e.g. complete blood analysis) respectively. The access control policy controlling the medical record could be (in pseudo SecPAL):

```
MedService says $x can read file://Alice/general if $x is a HealthActor
MedService says Alice can say0 $x can read file://Alice/confidential if
  $x is a HealthActor
MedService says MedicalAssociation can say0 $x is a HealthActor
MedicalAssociation says DrBob is a HealthActor
MedicalAssociation says HospitalX is a HealthActor
```

This policy mainly states that any medical service or person (HealthActor) can read general health information of Alice and that Alice can choose which medical service or person can read her confidential health data. For completeness, the policy also lists one medical service (HospitalX) and one medical person (DrBob), which is are certified by the medical association.

When Alice uses a service offered by HospitalX to monitor her health, she will get metadata specifying that full read access to some medical record is required. Metadata is provided to the delegation framework, which loads all matching[4] resource cards. In this example, we can assume that Alice is registered to only one medical record service and thus we can skip the step of selecting the card. In the

[3] Examples for health record service are http://healthvault.com/ and http://www.google.com/health

[4] The matching between the concrete resource and the description is out of the scope of this paper. Fore more information, see (Bussard et al. 2008).

delegation selector, Alice can verify the trustworthiness of the service offered by HospitalX and can see which rights will be delegated.

Based on the resource card, the corresponding plug-in is loaded. The SecPAL plug-in (as well as any other plug-in) implements the following interface:

```
void GrantAccess(AbstractDelegatee, ResourceCard, AbstractPriviledges)
void RevokeAccess(AbstractDelegatee, ResourceCard, AbstractPriviledges)
List<DelegationInformation> GetDelegationStatus(ResourceCard)
```

GrantAccess is called with three parameters. *AbstractDelegatee* mainly contains a credential of the delegatee, in this example a SecPAL Principal or an X.509 certificate. This credential is used for delegation, i.e. issuing a new SecPAL credential for the delegatee in this example. *ResourceCard* provides additional information about the resource to the plug-in, e.g. the resource address and the type of access control management. *AbstractPrivileges* contains the rights to be granted, i.e. read access to confidential part of a medical record. The plug-in will then handle the delegation by creating one or more new claims:

```
Alice says HospitalX can read file://Alice/confidential
```

The process of creating appropriated claims can be hard-coded in the plug-in and configured by the resource card or can rely on an abduction query to discover missing claims (Becker et al. 2009). Finally, plug-in creates a credential from the necessary claims and hands it to the delegatee.

Existing Plug-ins

STS Plug-in Security Token Services (STS) implement WS-Trust (Nadalin et al. 2008) interface to issue and validate credentials. First, to validate the model, a simple STS issuing SAML tokens based on an X.509 access control list was developed as resource guard. We created a plug-in to let the delegation framework deal with such delegation. When a delegation is invoked it takes the X.509 from the delegatee card and adds this to the STS's access control list. At invocation time of the resource the delegator use his private key to authenticate. In order to revoke a delegation the plug-in removes the X.509 from the STS's access control list.

Next, to go further in the integration of standards and legacy management application, we tested the delegation framework with the *.NET Access Control Service* of the *Azure Services Platform,*[5] a publicly-accessible STS offering a management interface. Access to the resource was protected with this STS and the plug-in had to be adapted to call the management API of this one. All necessary inputs (e.g. address of the management endpoint) are part of the resource card.

SecPAL Plug-in In order to validate the model, a prototype resource protected by SecPAL was implemented together with a SecPAL plug-in, which was used in the delegation framework. The SecPAL policy associated with the resource stated that someone owning a resource can delegate access to this resource. The delegator

[5] See http://www.microsoft.com/azure/accesscontrol.mspx

possesses a SecPAL token credential providing that she owns this resource. The delegation framework creates a new credential stating that the delegatee can access a subset of the resource for the required duration. The delegation credentials is provided to the delegatee, who in turn presents it when invoking the resource. A revocation of the access right is done by adding the issued delegation credential to a revocation list.

Google Calendar Plug-in We chose the Google Calendar API[6] to experiment with the integration of non-SOAP legacy management interfaces. The plug-in maps a delegation to management operations of the Google API. It essentially modifies rights of delegatees to read the calendar of the delegator.

Fingerprint Plug-in This plug-in is not related to service composition, but has been developed to look at resources that are not Web Services (SOAP or REST). In the demo application the plug-in was used to control access to a car. The delegation browser was used to specify that a delegatee 'Bob' can drive the car of the delegator for the next few days while 'Bob Junior' can open the doors but may not start the engine. The car application featured a fingerprint reader and individual access control lists for opening the doors and starting the engine. The plug-ins management operation consisted in uploading and removing the delegatees' fingerprint feature vector to the appropriate access control list.

Unified user experiences

In most cases, the delegatee is a service, e.g. mash-up or workflow, and thus does not directly involve delegatee-side users. In this article, we focus on the user experience of the delegator. When the delegatee is a human being, e.g. delegator Alice grants access rights to delegatee Bob, then the user experience of the delegatee becomes relevant. The delegatee may use an *Identity Selector* (e.g. CardSpace) when accessing delegator's resources. When the delegatee can delegate further, he acts as delegator and may thus use a *Delegation Selector*. Combining Identity Selector and Delegation Selector for such advance scenarios is an interesting option, but out of the scope of this article.

Figure 5 presents the *delegation selector*, which appears in delegatee-driven scenarios. In this sample, the user (delegator) is executing a sightseeing composite service (delegatee) that requires access to the user's real-time location (resource). First, the user (delegator) is asked for a trust decision regarding the composite service (delegatee). Her decision is based on PKI or reputation. Next, the delegation selector highlights the list of resources (1) that match the composite service requirements (first three cards are highlighted). In this example, the user is registered to three location services: a service providing the location from a GPS device, a service delivering a coarse-grained position based on the radio cell of user's mobile phone (selected), and a similar service that is protected by a SecPAL-based resource guard.

[6] See http://code.google.com/apis/calendar/

Before delegating, the user can see what rights she will grant (2). In this example the delegate may invoke the operation *GetLocation(Bob)* but not the operation *GetSpeed()*. Moreover the duration of the delegation is displayed. Finally, the user simply proceeds and delegation happens without having the user dealing with the underlying mechanisms.

A user experience has to deal with granting access but also with controlling ongoing delegations. Figure 6 shows the *Delegation Browser*, an experimental user interface to keep track of active delegations, to revoke them and to enable delegator-driven delegation. Each column corresponds to a delegatee card and each row is related to a resource card. The intersection between delegatee and resource provides information on ongoing delegation, which is represented with different icons. The browser shows details about a resource, delegatee, and delegation in the bottom part of the window. Resources as well as delegatees can be grouped enabling a more coarse grained view similar to the Expandable Grids by Reeder et al. (2008).

Related work

There is much prior art in the field of access control for web services. It can be distinguished in four groups. The first group focuses on access control for individual web services and strives to come up with policy languages, improvements in policy

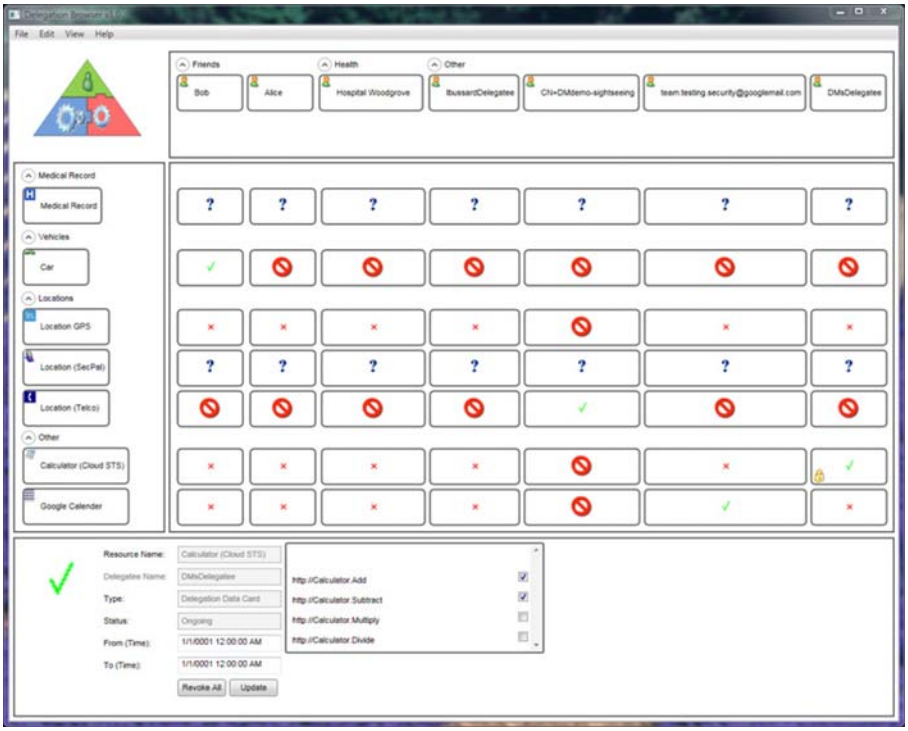

Fig. 6 Delegation Browser is an experimental user interface to keep control over ongoing delegations

decisions, or applies existing access control models to service oriented architectures.[7] The second group concentrates on access control for composite services in intra-organization applications. Hence, these publications explicitly or implicitly deal with scenarios where all services are inside a single trust domain. The third group of publications considers composite services distributed over multiple trust domains, but makes the implicit assumption or explicit proposal of a common authorization mechanism to circumvent the interoperability issue. The last group summarizes prior art that considers abstract management of access rights across multiple trust domains.

Policy languages

XACML (Moses 2005) stands for Extensible Access Control Markup Language. It is a declarative access control policy language and a processing model, describing how to interpret the policies. XACML has been widely adopted both by industry and academics. However, XACML does not support delegation of authority. Many publications (Lang et al. 2006; López et al. 2005; She et al. 2007; Yu 2006), which we will discuss below, use XACML as basis for their work, build extensions on top of it or provide interoperability.

SecPAL (Becker et al. 2007) is a declarative, logic-based security policy language developed to meet the access control requirements of large-scale Grid Computing Environments. It was designed to be comprehensive and provides a uniform mechanism for expressing trust relationships, authorization policies, delegation policies, identity and attribute assertions, capability assertions, revocations, and audit requirements.

AC for service compositions within a single trust domain

Robinson et al. (2006) present an approach to "automatically derive the minimal authorizations required for collaboration, as well as enable and disable the authorizations in a just-in-time manner that matches the control flow described in the choreography." They use it for "runtime management of authorization policies" of ad-hoc combined web-service, but the basic idea could be utilized as building block in the approach described in this article.

The approach presented by Mukkamala et al. (2006) supports real ad-hoc collaboration between services. The "dynamic coalition-based access control (DCBAC) model facilitates the formation of dynamic coalitions through the use of a registry service". A central registry which is also responsible for authorization decisions is used as trusted third party. A service registering itself at the registry may upload its individual service policy, e.g. expressed with WS-Policy, which makes it more suitable for single domain service compositions.

Wimmer et al. (2006) propose a formal model that allows the computation of a summarized policy from a set of individual policies. It computes the least required privileges for the execution of the composite application. They do not mention delegation or answer the question how the set of individual policies is compiled in the first place.

[7] As example, we present only two prominent policy languages.

AC for service compositions across different trust domain

The European project NextGrid[8] focused on service level agreements in cross-domain grid applications. NextGrid defines policy management operations that allow delegation in Grid service environments (NextGrid 2008). The Grid framework GRIA[9] implements this delegation concept as part of process-based access control (IT Innovation 2009). NextGrid defines standardized way to delegate access to resources. The delegation framework we're proposing could use the NextGrid protocol in a dedicated plug-in and thus opening the delegation framework for Grid services.

She et al. (She et al. 2007) recognize that access control for service composition across multiple trust domains is not solved yet and come to the conclusion that delegation of access rights will become an important mechanism for service composition. Their approach is a "delegation-based security model [which] provides the framework and defines the key processes to secure the web services through issuing, sending, confirming, negotiating, and revoking the delegation tokens." This in fact means that all web services participating in this composition have to agree on a single token-based authorization mechanism, in She's case it is an extension of XACML policies. We think that this is not practicable and that a delegation solution has to integrate a large diversity of authorization mechanisms.

A case study how to integrate web services which are protected by XACML, SAML and PERMIS and that span across multiple trust domains is presented by López et al. (2005). They follow the same rational we give in this article: "standards for authorization systems are less widely adopted and accepted, and tend to work only within homogeneous systems". López et al. base their integration on a previously published Credential Conversion System (Cánovas et al. 2004). The approach makes use of role-based access control, which in turn "leads to the definition of loosely coupled multi-domain environments, where a predefined set of role mappings to mediate inter-domain accesses is defined." However, the solution requires a high level of administration which makes it not suitable for ad-hoc delegation.

Freudenthal et al. (2002) propose a model for distributed role-based access control (dRBAC) in "coalition environments". They describe a distributed network of collections of delegation credentials, which they call 'wallets'. "A trust-sensitive system resource can query a wallet for proofs authorizing whether a requested access is permitted." The whole approach is similar to the SecPAL policy language (Becker et al. 2007), which addresses the same question by means of a formal language whereas dRBAC utilizes an infrastructure.

OAuth (OAuth Core Workgroup 2007) is an open source protocol that allows users to delegate access to their resource to a delegatee.[10] It explicitly covers the situation that the delegatee is an application as well. The protocol keeps the delegator's identity and credentials confidential, which in fact allows anonymous delegation. OAuth defines an access control and management mechanism, but does

[8] http://www.nextgrid.org
[9] http://www.gria.org
[10] OAuth refers to resource as *protected resource* and to delegatee as *consumer*.

not take into account the variety of existing solutions. Although it is based on well established standards it needs to be widely adopted in order to have an impact and thus adds to the diversity we described in the introduction.

Abstract management of access rights

Yu (2006) proposes a "reusable access control layer, which is separate from the web services themselves. All web service requests pass through the authorization layer. An authorization decision is made and passed onto the web service." Apparently the goal of this work is to have a central policy decision point which additionally features an inference engine for policy decision. Although the authors touch extensibility for other access control mechanisms, the nature of a central decision point prevents this solution from being used in ad hoc service compositions.

Lang et al. (2006) present an approach for "multipolicy authorization" that allows to use XACML and SAML in Globus Toolkit release 4, a middleware for computing Grids (Foster et al., 2001). They abstracted a policy decision point (PDP) with an object-oriented interface, which supports an abstract operation 'canAccess()'. "Each specific policy is a subclass of the PDP abstraction, which implements the common interface inherited from PDP with its own policy and evaluation mechanism." Our approach also uses an abstraction for authorization management, but we offer an improved flexibility by adding new resource cards and plug-ins.

Windows Live ID Delegated Authentication (Windows Live 2008) is "a way to permit access to personal information, but with more precise control over access and usage permissions than the [...] generally bad practice of handing over your account credentials to another Web site." This approach fully supports our point of combining services from multiple trust domains. An interesting characteristic is to split the delegation in two phases, the consent phase and the delegation phase, which allows that the delegator needs to be online only for consenting. Windows Live ID Delegated Authentication defines an interesting delegation mechanism, but does not take into account the variety of existing solutions.

One of our former papers (Bussard et al. 2008) already motivates why abstract delegation is required in service composition and how this concept could be used with the SCENE infrastructure, a platform that supports the development and execution of self-adaptable service centric systems. This former paper describes just the high-level concepts and use-cases, but lacks deeper technical content. In this article we explicitly focus on the technical details of abstract delegation, i.e. delegation plug-ins, support for specific types of delegation, and user experience. Moreover we present results from our validation implementation.

Concluding we can say that we are not aware of any prior art that allows convenient delegation in multi-domain service compositions which regards the existing variety of access control mechanisms.

Discussion

The proposed solution has a number of advantages over the current state of the art. First of all, it solves the problem to allow delegation in multi-domain service

compositions. Thus it removes the roadblock for service compositions that we identified in the introduction as essential problem in today's technology. It was important to find a solution that does not define yet another access control mechanism or policy language, but leverages multiple existing access control mechanisms. The variety of plug-in we tested shows that the abstract delegation works in wide range of technical scenarios. Moreover the solution allows the user to manage and supervise delegation actions. A unique user interface controls all delegations regardless of the underlying access control mechanism. The delegation browser allows the grouping of resources and delegatees and thus provides a quick general overview to the user. At the same time, it gives detailed information on every single delegation and enables the user to grant and revoke rights in an easy and unified way. We see the abstraction of access control and delegation mechanisms as well as the unified user experience as the main contribution of this article.

The approach proposed in this article can be deployed without changing an existing security infrastructure (STS, XACML PDP, etc.) or the resource. However, end-users (delgators) require the delegation framework and its user interfaces. Depending on the trust model, these components could be hosted in the cloud and accessed with a web browser. At delegatee-side the composite applications must be "delegation-aware" to benefit from the delegation abstraction.

The approach presented in this article does not impact existing Identity Management Systems and thus does not introduce new scalability issues. However, simplifying the way of querying authorization services and resources as well as making users aware of delegation may lead to more queries than we see today. The main performance issues we noticed during this work were on client-side when updating simultaneously the status of tens of resources owned by a user (calendar, location service, medical record, etc.). Running each query in a dedicated thread enables update in a acceptable time (few seconds). Apart the framework itself, the current implementation of the user experience, esp. the Delegation Browser, would not be convenient with hundreds of resources. Better grouping mechanisms would be necessary.

Certainly, our research is far from being complete. We see a number of shortcomings that could be taken up and addressed by the research community. First, the configuration of access control mechanisms becomes difficult in complex scenarios where overlapping access rights are granted and revoked multiple times. Second, we presented a way to keep track of ongoing delegations with the delegation browser. It shows the current status of delegation, but does not keep track of the history of delegations, which is especially important in case of dynamic issuance of new claims, e.g. with SecPAL (Becker et al. 2007). Finally, from an ergonomic point of view the CardSpace-like delegation selector was just an experiment. More work on human computer interaction is necessary to understand what kind of interface is most suitable.

Conclusion

We describe a delegation framework which allows delegation of access rights in multi-domain service compositions. We explained why this is an essential

requirement to improve the current state of the art. We describe a delegation framework that is able to delegate access rights to a resource protected by an existing access control mechanism. The two main advantages of the solution are a) that it supports multiple existing access control solutions in a generic way and b) that all delegations are performed and monitored trough a unified API and user interface, regardless of the underlying access control mechanism. The article is completed by an extended literature research revealed that this delegation capability is still lacking although it is mentioned by some publications.

Acknowledgements Parts of the implementation work were carried out under the auspices of the European Commission in course of IST FP6 project SeCSE.[11] The authors express their thank to Joris Claessens, Christian Geuer-Pollmann, Olivier Nano, Marcel Tilly, and Muhammad Ali from European Microsoft Innovation Center for valuable discussions on the concepts of this delegation abstraction. The technical details and opinions expressed in this article are the authors' point of view and not necessarily the one of Microsoft Corp. or the project participants in SeCSE.

References

Becker MY, Fournet C, Gordon AD. Design and semantics of a decentralized authorization language. 20th IEEE Computer Security Foundations Symposium (CSF). 2007.

Becker MY, Mackay JF, Dillaway B. Abductive authorization credential gathering. IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY). 2009.

Bussard L, Di Nitto E, Nano A, Nano O, Ripa G. An approach to identity management for service centric systems. ServiceWave 2008, Madrid, Spain. 2008.

Cánovas O, López G, Gómez-Skarmeta AF. Acredential conversion service for saml-based scenarios. In: Public key infrastructure, volume 3093 of Lecture Notes in Computer Science (LNCS). Berlin/ Heidelberg: Springer; 2004. p. 297-305. ISBN 978-3-540-22216-3.

Fitzpatrick B, Recordon D, Hardt D, Bufu J, Hoyt J. Openid authentication 2.0 -final. Specification, OpenID Community, Dec. 2007. http://openid.net/specs/openid-authentication-2_0.html.

Foster I, Kesselman C, Tuecke S. The anatomy of the grid: enabling scalable virtual organizations. International Journal of High Performance Computing Applications. 2001;15(3):200–22.

Freudenthal E, Pesin T, Port L, Keenan E, Karamcheti V. dRBAC: distributed role-based access control for dynamic coalition environments. In: Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS '02). Washington, DC: IEEE Computer Society; 2002. p. 411–20.

IT Innovation. PBAC 2 (Process-Based Access Control) Manual. 2009.

Lang B, Foster I, Siebenlist F, Ananthakrishnan R, Freeman T. A multipolicy authorization framework for grid security. In: Fifth IEEE International Symposium on Network Computing and Applications. IEEE Press; 2006. p. 269–72.

López G, Cánovas O, Gómez-Skarmeta AF, Otenko S, Chadwick DW. Public key infrastructure, volume 3545 of Lecture Notes in Computer Science (LNCS), chapter A heterogeneous network access service based on PERMIS and SAML. Springer; 2005. p. 55–72. ISBN 978-3-540-28062-0.

Moses T. OASIS eXtensible Access Control Markup Language (XACML) Version 2.0. OASIS Standard oasis-access control-xacml-2.0-core-spec-os, OASIS. 2005.

Mukkamala R, Atluri V, Warner J, Abbadasari R. A distributed coalition service registry for ad-hoc dynamic coalitions: a service-oriented approach. In: Damiani and Liu. Data and applications security XX, volume 4127 of Lecture Notes in Computer Science (LNCS). Berlin/Heidelberg: Springer; 2006, p. 209–223. ISBN 978-3-540-36796-3.

[11] SeCSE project website is available at http://www.secse-project.eu/.

Nadalin A, Goodner M, Gudgin M, Barbir A, Granqvist H. OASIS WS-Trust 1.4. Specification Version 1.4, OASIS, Feb. 2008. Currently in draft status, refer to version 1.3 for latest approved version.

NextGrid. NextGRID Security Profile. 2008.

OAuth Core Workgroup. OAuth Core 1.0. Technical report, 2007.

Patil V, Mei A, Mancini LV. Addressing interoperability issues in access control models. In: Proceedings of the 2nd ACM symposium on information, computer and communications security. New York: ACM; 2007. p. 389–91. ISBN 1-59593-574-6.

Reeder RW, Bauer L, Cranor LF, Reiter MK, Bacon K, How K, et al. Expandable grids for visualizing and authoring computer security policies. ACM SIGCHI Conference on Human Factors in Computing Systems (CHI '08). 2008.

Robinson P, Kerschbaum F, Schaad A. From business process choreography to authorization policies. In: Damiani and Liu. Data and applications security XX, volume 4127 of Lecture Notes in Computer Science (LNCS). Berlin/Heidelberg: Springer; 2006, p. 297–309. ISBN 978-3-540-36796-3.

SeCSE Consortium. Design of the 3rd version of the SeCSE delivery platform (focused on IdM). Public report A4.D19, SeCSE Project. 2008.

She W, Thuraisingham B, Yen I-L. Delegation-based security model for web services. In: Proceedings of 10th IEEE High Assurance Systems Engineering Symposium (HASE '07). IEEE Computer Society; 2007. p. 82–91. ISBN:978-0-7695-3043-7.

Wimmer M, Kemper A, Rits M, Lotz V. Consolidating the access control of composite applications and workflows. In: Damiani and Liu. Data and applications security XX, volume 4127 of Lecture Notes in Computer Science (LNCS). Berlin/Heidelberg: Springer; 2006, p. 44–59. ISBN 978-3-540-36796-3.

Windows Live. Understanding Windows Live delegated authentication. White paper, Microsoft Corporation, Feb. 2008. http://msdn2.microsoft.com/en-us/library/cc287613.aspx.

Yu WD. An intelligent access control for web services based on service oriented architecture platform. In: Proceedings of the The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06). IEEE Computer Society; 2006. p. 190–98. ISBN:0-7695-2560-1.