2010 International Conference on Availability, Reliability and Security

A Semantic Security Architecture for Web Services – the Access-eGov Solution

Stefan Dürbeck, Christoph Fritsch, Günther Pernul and Rolf Schillinger
Department of Information Systems
University of Regensburg
93053 Regensburg, Germany
Email: {stefan.duerbeck, christoph.fritsch, guenther.pernul, rolf.schillinger}@wiwi.uni-regensburg.de

***Abstract*—The shift from mere service-oriented architectures (SOA) to semantically enriched approaches is especially being forced in multi-domain environments that the public sector in the European Union is an example for. The security aspect is lagging behind its possibilities, and new access control approaches native to the semantic environment need to be applied. Based on architectural research work conducted within the EU-funded research project Access-eGov, we outline our implementation of a semantic security architecture for web services by using industry-standard technologies and combining them with semantic enhancements.**

***Index Terms*—XACML, semantic security infrastructure, eGov**

I. Introduction

Nowadays, public administrations all over the world are being confronted with the ever growing challenge to deliver their services electronically and, especially in the European Union, are required by the so-called Services Directive[1] to do this across their own legal boundaries.

Challenges to this vision mainly arise from the semantic gap between the service descriptions, resulting in high entry thresholds to implement those collaborative systems. In enabling semantically enriched service-oriented architectures (SSOA), Access-eGov saw its main field of research.

Restricting access to valuable service resources only after trustful authorization was seen as a crucial part to heighten the overall security of the semantic service-oriented system and the reliability in general [2]. Our semantic security architecture for service-oriented systems based on the attribute-based access control paradigm [3] has already been described in [4], [5], [6] and has now successfully been implemented.

In the following chapters, we describe the technological building blocks for implementing authorization and access control in service-oriented architectures and also introduce the related key technologies that already exist in this field of research (chapter II). Chapter III gives an overview of our approach to securing a service-oriented architecture and shows architectural alternatives how to introduce semantically-enabled access authorization here. Selected aspects of the respective implementation are being described in chapter IV, before desirable future research activities in this field are being summed up in chapter V.

II. Building Blocks and Related Work

A. Security Building Blocks

The security architecture for the Access-eGov project does not aim at inventing new authentication and authorization standards and protocols. Instead, our security infrastructure is based on a number of existing and proven standards. However, no single standard allows realizing the security infrastructure as intended as each one is focused on a very specific function. In the following we therefore shortly lay out the basic technical principles.

1) WS-Security: WS-Security [7] is an open standard that specifies how security related meta-data should be incorporated into SOAP messages. WS-Security does however not define security models, mechanisms or technologies but rather defines how existing approaches should be applied to SOAP messages to ensure interoperability among different implementations and languages. For that purpose WS-Security defines several basic elements for the SOAP headers to hold security information. Several additional profiles for security tokens have been specified so far, such as the *UsernameToken Profile* [8], or the *X.509Token Profile* [9]. For the security infrastructure of Access-eGov the *SAMLToken Profile* [10] is the most relevant one as it allows integrating SAML assertions in the SOAP header. The WS-Security standard and its token profiles are currently supported by several web service frameworks such as Apache Axis 2 and its security module Rampart, Sun Metro and its security component WSIT or Apache CXF.

2) SAML: The Security Assertion Markup Language (SAML) [11] is an extensible XML-based specification standardized by the OASIS to exchange authentication and authorization data between identity providers (IdP) and service providers (SP). SAML is used in many SSO projects such as the Shibboleth project or the Liberty Alliance and is therefore our choice. For exchanging information, SAML defines a number of request/response protocols and the data type of *SAML Assertions*. A SAML Assertion is information container which can carry one or several *SAML Statements* about a subject, typically a user. SAML Assertions are issued by SAML authorities, e.g. a user's IdP, and are used by SPs to grant or reject access to protected resources based on the contained statements. Due to the available extension library to support the SAML profile for XACML (see section II-A4) we decided

978-0-7695-3965-2/10 $26.00 © 2010 IEEE
DOI 10.1109/ARES.2010.117

to build our security architecture based on OpenSAML[1].

3) XACML & WS-XACML: Before even being able to pass security tokens around, the system needs to arrive at an access control decision first. For this purpose our security infrastructure roughly follows the XACML specification [12]. The XACML specification is an OASIS standard which basically comprises a XML-based access control policy language and a architecture recommendation for authorization infrastructures. Specifically, the XACML architecture specifies the implementation of a Policy Enforcement Point (PEP), a Policy Administration Point (PAP), a Policy Decision Point (PDP), a Policy Information Point (PIP), and a Context Handler. Each of these actors is devoted to one specific task in the access control process.

The *Web Service Profile of XACML* (WS-XACML) [13] specification defines means for the application of XACML in a SOA. While WS-Security (see section II-A1) defines how several kinds of security tokens should be integrated into SOAP messages, WS-XACML specifies one of these tokens, the *Authorization Token*, in more detail. The Authorization Token allows transferring an access decision between PDP and PEP. This enables the PDP as a trusted third party to make access decisions on behalf of the SP.

Currently available implementations of the XACML specification are for example OpenXACML[2], enterprise-java-xacml from Google code[3], HERAS XACML[4], JBoss XACML[5], and Sun's XACML[6]. At the time we started implementation we selected Sun's XACML implementation as it was the most mature one available.

4) SAML Profile for XACML: Neither XACML nor WS-XACML specify a messaging protocol for communication between the different nodes of the architecture but rather rely on SAML to transfer the specified security information. The SAML 2.0 profile for XACML v2.0 [14] bridges the gap between SAML and XACML. It clearly specifies how XACML messages are to be transfered via the SAML protocol and how XACML attributes are represented in SAML.

At the time we started implementing the security infrastructure, the SAML profile for XACML was implemented by the "OpenSAML2.0 Extension Library to Support SAML2.0 profile of XACML2.0"[7] and integrated well with OpenSAML. In the meantime it is already fully integrated into OpenSAML.

5) WS-Security SAML Token Profile: The last missing piece for implementing our security infrastructure is the WS-Security SAML Token Profile. SAML assertions can be used within SOAP messages to carry authentication and authorization information [15]. Although WS-Security defines a security header for SOAP messages to hold such data, it does not define its format and structure. This profile therefore clearly defines how SAML assertions should be used for this purpose with the WS-Security framework to avoid incompatibilities between different implementations.

[1]http://www.opensaml.org
[2]http://www.openxacml.org
[3]http://code.google.com/p/enterprise-java-xacml
[4]http://www.herasaf.org/components/xacml-implementation.html
[5]http://www.jboss.org/jbosssecurity/downloads/JBoss\%20XACML/
[6]http://sunxacml.sourceforge.net
[7]http://www.bccs.uib.no/~hakont/SAMLXACMLExtension

B. Semantic Building Blocks

Semantic SOAs need a sophisticated back end, mainly consisting of an entity representing world, domain, and web service specific knowledge (usually an ontology), a language for serializing this knowledge for different on-the-wire protocols, and an associated set of components and tools to query, store and transform it. In our approach we use the Web Service Modeling Ontology (WSMO) [16], the Web Service Modeling Language (WSML) [17] and an adopted and extended version of the accompanying tool set:

1) Web Service Modeling Ontology: In sharp contrast to other approaches such as OWL-S [18], the WSMO development team chose not to rely on existing technology but started the development of an ontology for semantic web services from scratch, creating an attractive conceptual model with only four main elements: Ontology, Web Service, Goal and Mediator.

2) Web Service Modeling Language (WSML): WSMO constructs are formalized into WSML, a XML-based language that is available in four variants with differing degrees of complexity and different logic formalism roots.

3) SAWSDL: A web service grounding describes the actual web service endpoints, protocols, and the messages exchanged to use them. Conventional SOAs keep this information in plain WSDL documents which have to be prepared for usage in semantically enhanced scenarios. Of the approaches explained in [19], we chose the backward compatible SAWSDL [20] approach for its flexibility and simplicity. One element, the model reference, links functions of a web service (described in WSDL) to their semantic representation. Elements for lifting and lowering, the processes of transforming data from the low-level web service to the semantic world and vice versa, complete the necessary information and allow direct execution of a conventional web service. Currently the method of choice for lifting and lowering translations is to use XSLT where possible to quickly define effective transformation rules.

4) Additional Components: In order to employ our security infrastructure, we need to first encode the security services into WSMO entities. A web service instance that offers the capability authentication has to be annotated with this capability via a user interface. Web services in need of authentication need to be annotated with a complementary goal of the same name. A *resolver component* is used to find those services capable of fulfilling a specific goal. Execution of those services is triggered and monitored by an *invoker component*.

5) Process Model: The original WSMO process model is based on abstract state machines. For a number of reasons given in [21] and [6] however, Access-eGov has defined its own process model. It builds upon a hierarchical organization of life events, associated goals and sub-goals and a rule-based way of describing the interfaces.

III. A Semantic Security Architecture for Web Services

First ideas and the preliminary overall architecture of our security infrastructure for Access-eGov have already been presented in [4] and [5]. Similar approaches have as well been identified by [22], [23], [24]. Therefore, this section focuses on the differences and adoptions we performed for real-world deployment. The presented architecture and data flow implement all requirements gathered in [25], [26], [2].

The XACML standard defines a basic authorization infrastructure that is generic enough to implement the ABAC authorization model as proposed in the Access-eGov project. We therefore based our security infrastructure (see Fig. 1) on the XACML approach (see section II-A3).

A. Building Blocks and Data Flow

The *Access Requester* is the central component of our security infrastructure (see Fig. 1). In our scenario the user interacts with the Access-eGov portal via a web browser (*step 0*). Therefore the Access Requester is implemented as part of the Access-eGov portal and acts on behalf of the user. To interact with the Access-eGov portal, a user has to be authenticated. The focus of this paper however is on the authorization process. We therefore presume successful user authentication which can easily be handled by many widespread authentication mechanisms [27]. In *(1)* the Access Requester retrieves the access policy of the resource from the PAP by sending a `XACMLPolicyQuery` to the PAP.

The PAP returns the requested policy within a `XACMLPolicyStatement` *(2)* from which the Access Requester in turn retrieves all sets of user attributes which potentially result in a positive authorization decision.

In *(3)* the Access Requester retrieves the user attribute values for one of the attribute sets of *(2)* via a `SAMLAttribute-Query` from the PIP. The *PIP* is the source of all kinds of attributes. Our architecture does not imply a single, central PIP but allows for different PIPs for different kinds of attributes. We suggest to have a dedicated PIP for user attributes as they are considered especially worth protecting. Retrieval of user attributes from the PIP is protected and likewise requires authorization. This is out of scope for this paper but [28] for example presents a recursive approach to use XACML for this kind of authorization as well.

As the PIP and the SP may employ different attribute naming schemes, not all requested attributes may be known to the PIP. Therefore the PIP optionally sends a list of unknown attribute identifiers to the Inference Engine *(4)*. The *OAP* is an optional component that manages all ontologies employed by the different SPs within the Access-eGov system and keeps them ready for the Inference Engine. The Inference Engine retrieves the ontologies used by the PIP and the SP respectively from the OAP and mediates between different attribute identifiers. In *(5)* it returns a list of alternative user attribute identifiers which are known to the PIP and semantically identical to the ones requested in the access policy of the resource. Finally, the PIP returns the requested user attributes within a `SAMLAttributeStatement` *(6)* to the Access Requester. In case the PIP does not return attribute values for all requested user attributes (some of them may be unknown), the Access Requester repeats *(3)* with a different attribute set.

In *step (7)* the Access Requester optionally employs a *PDPSelector*. The PDPSelector is not defined in the XACML standard and is an optional component of our security infrastructure. Users have to make available their attributes to the PDP to access restricted resources but might as well restrict disclosure of their attributes to selected PDPs. The PDPSelector is in charge of discovering a PDP that both, the user and the SP, have trust in and return it in *(8)* to the Access Requester.

As the Access Requester now gathered all required user attribute values and an applicable PDP, it retrieves an authorization decision from the selected PDP via a `XACMLAuthzDecisionQuery` *(9)*.

The *PDP* is the most important component of our security infrastructure. It finally decides whether a user is allowed to access a given resource or not. It therefore retrieves the policy of the requested resource via a `XACMLPolicyQuery` *(10)* from the PAP, which returns it in form of a `XACMLPolicyStatement` *(11)*. Hence, *(10)* and *(11)* are identical to *(1)* and *(2)* respectively.

To finally reach an authorization decision, several entity, subject, resource, action, and environment attributes may be required. As the `XACMLAuthzDecisionQuery` only contains the required user attributes, the PDP queries the PIP for the missing entity, resource, action, and environment attributes via an `SAMLAttributeQuery` *(12)*. Similar to *(3)*, there may be different naming schemes for the missing attributes which are resolved by the Inference Engine (*(13)* and *(14)*). Finally the PIP returns the required attribute values within a `SAMLAttributeStatement` *(15)*.

The PDP now holds all required information to evaluate the access policy and returns the resulting authorization decision back to the Access Requester in form of a `XACMLAuthzDecisionStatment` *(16)*.

The Access Requester finally embeds the received `XACML-AuthzDecisionStatment` into the SOAP header of the resource request message and delivers it to the *Resource* (*(17a)* and *(17b)*). More generally, the PEP is an application-specific software component that is physically enforcing access control to a resource. The PDP intercepts all communication with the protected service, analyses the embedded `XACMLAuthz-DecisionStatment`, and according to its content, permits or denies access to the protected service. The Resource finally does not have to care about security, authentication and access control as these functions are performed by the PEP, that is put in front of it. The Resource simply returns the results of its business logic back to the Access Requester (*(18)*) which in *(19)* presents the result to the user.

IV. Implementation

In the Access-eGov public service pilot case, we focused on implementing the necessary core components as depicted in Fig. 1 and substituted module functionality as needed.

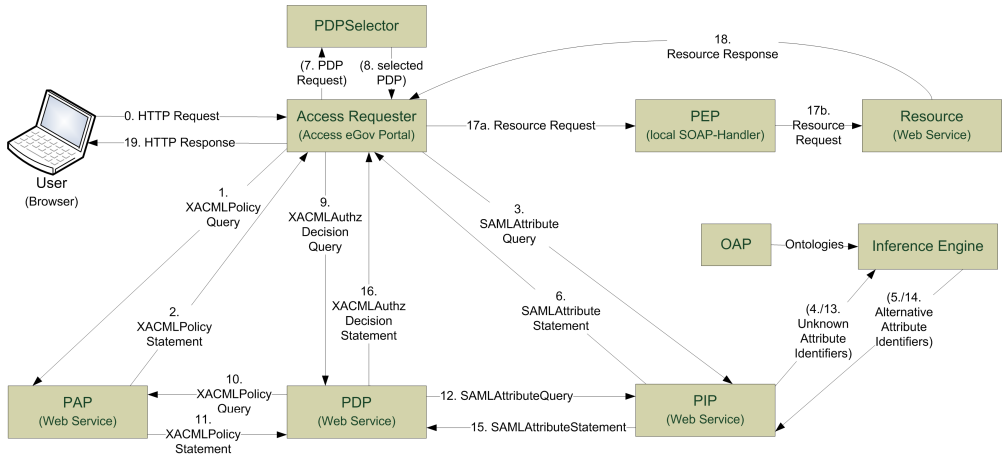

Fig. 1. Security Architecture in Access-eGov

The Access-eGov service landscape consists of several dozen public community administrations offering administrative (web) services around the life event ”registering for marriage” and has been set up in collaboration with the German State Government of Schleswig-Holstein and its Ministry of Finance in Kiel.

In our fictitious case presented here, the resource whose functionality shall be protected against misuse, is the reservation web service of a sight-seeing boat which can be booked for the marriage ceremony and the related function. The ”LoveBoatKiel” web service offers the functions `reserveForMarriageFunction` and `reserveOptionalCaptainsDinner` to allow for an optional captain's dinner as well. In order not to distract the reader with legal peculiarities of marrying in Germany, we simply assume the web service is only requesting the booking person to prove being of legal age (i.e. to indirectly prove he/she can book the boat for his/her own marriage function) before a reservation can be carried out.

A. Realizing XAMCL Entities as Web Services for Loose Coupling

In addition to the service landscape that was already present across the Access-eGov small public service world, the before-mentioned XACML modules PIP, PDP and PAP were implemented as separate web service components which are capable of being deployed independently from one another. This way, the security components can be brokered just as other functional services across the public service network.

The only exception to this service-oriented paradigm in our implementation is the PEP, which is transparently placed in front of the respective service provider. In order to guard the LoveBoatKiel service with a PEP, the PEP is simply put in front of the marriage reservation service via a SOAP message handler chain that is configured locally at service provider. This reduces the integration of access control to an administrative minimum and there is no further need for the servicing resource to know more about security.

The SOAP request with the access token in its message header and the functional parameter in its body, is handed over to the web service resource (see `(17a)` in Fig. 1).

This way, when requesting the protected resource, a chain of services is being called prior to the actual web service, with the PEP expecting an access token issued to grant access to the service and wrapped in the header of the incoming SOAP messages.

The web service method `reserveForMarriage-Function` is only executed when the SOAP handler chain does not block, i.e. the PEP accepts the access token of the Access Requester and hands over the data flow to the actual web service resource (see `(17b)` in Fig. 1). A SOAP message is generated with a reservation number to acknowledge receipt for the marriage function planning (see `(18)` in Fig. 1). In case the web service resource is accessed fraudulently or directly (without any access token at all in the SOAP header), the handler chain blocks further execution just at the PEP and forces the local SOAP messaging stack to reply with an error code accordingly.

B. Data Exchange between XACML Web Service Instances

Context Syntax is used in order to describe the language constructs for XACML Request and Response Contexts, i.e. data container formats for authorization requests and decision statements.

An XACML Request Context is a data container that an Access Requester builds to send it as `XACMLAuthzDecisionQuery` to the PDP. Each one of the four elements `subject`, `resource`, `action` and `environment` is optional, with every single one being solely defined by its attributes. Thus, a resource is not only presented by its actual URI, but the URI is one of many attributes specifying the resource. An attribute therefore can be any quality of an entity that can be described using XML. In our implementation the Request Context only contains user attributes (i.e. element subject) that a PDP may need to come to an authorization decision. The PDP retrieves all remaining attributes directly from the PIP.

C. Integration into the Access-eGov Platform

In detail, an exemplary data flow through the implemented security architecture starts at the user client, in Access-eGov a special portal application called "Personal Assistant Client", which has been publicly tested in early 2009 and is playing the role of the Access Requester as mentioned in chapter III-A. We assume a user has already electronically registered for marriage at a registrar's office in Kiel and is now opting for the captain's own optional service in order to reserve the boat as marriage venue. The Personal Assistant Client as Access Requester has been made aware of the access restriction to the LoveBoatKiel service and is now triggering the depicted security activities chain to obtain an access token for the web service to make a reservation.

Depending on the requester attributes that a service requires from a service user, the Personal Assistant Client asks the PIP to retrieve some or all suitable user attributes from the user's attribute repository on behalf of the service requester.

In our case, the PIP is part of the user management system with access to the user accounts, where all attributes are stored. The PIP returns SAML-compliant attributes, e.g. the user attribute values for legal age (see Listing 1) after successful authentication with the user's own Access-eGov platform SSO-credentials.

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:PIPAttributeResponse
     xmlns:ns2="http://pip.security.accessegov.org/">
    <ns2:Attribute AttributeId="urn:accessegov:attribute:legal-age"
     DataType="http://www.w3.org/2001/XMLSchema#string"
     IssueInstant="2009-04-17T11:40:45-05:00"
     Issuer="City_of_Kiel">
      <AttributeValue>true</AttributeValue>
    </ns2:Attribute>
    </ns2:PIPAttributeResponse>
  </S:Body>
</S:Envelope>
```

Listing 1. SAML attribute statement expressing legal age

The `legal Age` attribute was used in this scenario to guarantee that the target resource is basically protected against misuse by unintended service requesters. But other more meaningful attribute types can also be envisioned (e.g. user group attributes allowing access with certain membership entitlement only, etc.).

D. Semantically enhanced Integration of Security Services

Security services in a SSOA can, due to the flexibility of the underlying platform, be integrated using a number of different approaches of which we will describe a selection of the most important ones. The choreography – specified according to the Access-eGov process model – of the LoveBoatKiel web service as shown in Listing 2 serves as the basis, on which we will describe the different methods. This code conveys the following information:

- Variables
  `?input` and `?output` are defined to be shared variables during execution.
- Perform receive
  Instructs the platform to expect a semantic object of type `LoveBoatServiceInput` as the input to the service (modeling is always done from the service's point of view). Furthermore defines the URI at which the web service is reachable (`aeg#endpointGrounding`) and the lifting and lowering schemes of SAWSDL (see section II-B3).
- Perform send
  Instructs the platform to generate a LoveBoatServiceOutput semantic object and bind it to the variable `?output`.

```
1 webService LoveBoatService
2 [...]
3 interface LoveBoatServiceInterface
4  choreography LoveBoatServiceChoreography
5   sharedVariables { ?input, ?output}
6    transitionRules
7     perform receive ?input memberOf LoveBoatServiceInput.
8      nfp
9       aeg#endpointGrounding hasValue
10       _grounding("http://localhost:9998/alb/LoveBoatService?wsdl")
11      sawsdl#loweringSchemaMapping hasValue
12       _iri("http://localhost:8080/LoveBoatService_Lowering.xslt")
13      sawsdl#liftingSchemaMapping hasValue
14       _iri("http://localhost:8080/LoveBoatService__Lifting.xslt")
15     endnfp
16    perform send ?output memberOf LoveBoatServiceOutput.
17     nfp
18     endnfp
```

Listing 2. WSML code of an example web service

We can currently conceive of four architectural options for realizing such a security architecture for web services, namely static service authorization, and three fully semantic authorization types either through a sub-goal, through a Life Event or through modeling the XACML architecture. We implemented the static approach in the first place and will furthermore enhance this implementation applying the approach to "Full semantic Authorization through a sub-goal".

Static Service Authorization (SSA) is the most basic form of integrating security into a SSOA. It does not involve any form of semantic matching but features a binding to a service statically defined at annotation time. When implementing this approach, the interface in Listing 2 is prepended with an additional `perform receive` block constructed like the one on lines 9 - 14 and featuring a grounding to an authorization web service. Additionally a new variable has to be introduced to carry the security token:

1) Lowering (Authorization)
   The platform instantiates a semantic object comprising all necessary information to feed the authorization process shown in chapter III. An XSL transformation serializes this still semantic object and injects the required constituents into a SOAP message directed at the annotated web service.
2) Lifting (Authorization)
   The authorization step is performed and if the outcome of the decision is positive, the service returns a SAML token in the resulting SOAP message.
3) Mediation
   Due to peculiarities of the used Access-eGov platform, passing data from one invocation of a service to an invocation of another service involves an Access-eGov specific, so-called ppMediator which transports the variables between service invocations.
4) Lowering (Requesting service)
   Like the previous lowering step, the SAML token is injected into the SOAP header.
5) Lifting (Requesting service)
   This step transforms the web service output (in our case the result of the boat reservation order) back into the

semantic domain by extracting data from the response and instantiating new semantic objects.

As laid out in the previous chapter, the semantically enriched service-oriented security architecture can be deployed in separate self-contained modules within Access-eGov. Those modules are offering their respective functionality in accordance with the XACML standard for attribute based access control.

V. Future work

So far we only implemented the Static Service Authorization approach as the closest for semantic enhancement.

Ignoring the other architectural approaches to a semantic security architecture for web services, there are still a number of future actions that are worth examining, like the use of XMLSignatures. However, digital signature especially of the XACMLAuthzDecisionStatments from the PDP which contain the Request and Response Contexts is required for real-world application. For future versions of our implementation this will be dealt with by either employing SUN's WSIT or Apache's Rampart which allow for easy integration of WS-Security related capabilities such as XMLSignature and XMLEncryption. Also, the current implementation as described herein is still making use of an optional 3rd party SAML-XACML-binding profile. In the meantime, this binding has already been integrated into the latest version of the OpenSAML-API.

Another field for future research activity is the semantic layer of our implementation. All semantic annotations have been manually coded in WSML due to a lack of fully automated tools to support this activity, although a basic editor already exists. More relevant for the security aspect of our approach is the lack of any ontology for security constructs. Up to now, there is no sufficient ontology, or even controlled vocabulary of security terms publicly available for use in semantic access control systems for web services. For instance, authorization capabilities could then be described using a security ontology, with the authorization web services being semantically annotated for real-time service brokering. This will be a research topic of paramount nature to research into.

Acknowledgment

We would like to thank our project partners for helpful comments and stimulating discussions. This work was achieved within the Access-eGov project, which was supported by the European Union under the IST Programme, contract No. FP6-2004-27020. The content of this publication is the sole responsibility of the authors, and in no way represents the view of the European Commission or its related bodies.

References

[1] European Commission, "DIRECTIVE 2006/123/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 12 December 2006 on services in the internal market"," *Official Journal of the European Union*, 2006.

[2] S. Dürbeck, R. Schillinger, and J. Kolter, "Security Requirements for a Semantic Service-oriented Architecture," *Proc. of the Second International Conference on Availability, Reliability and Security (ARES 2007)*, pp. 366–373, 2007.

[3] T. Priebe, W. Dobmeier, B. Muschall, and G. Pernul, "ABAC - Ein Referenzmodell für attributbasierte Zugriffskontrolle," *Proceedings of the 2nd Jahrestagung Fachbereich Sicherheit der Gesellschaft für Informatik (Sicherheit 2005)*, 2005.

[4] J. Kolter, R. Schillinger, and G. Pernul, "A Privacy-Enhanced Attribute-Based Access Control System," in *Proc. of the 21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, pp. 129–143, 2007.

[5] J. Kolter, R. Schillinger, and G. Pernul, "Building a Distributed Semantic-aware Security Architecture," *Proc. of the IFIP TC 11 22nd International Information Security Conference (SEC 2007)*, pp. 397–408, 2007.

[6] R. Schillinger, S. Dürbeck, and P. Bednar, "Access-eGov - a real-world Semantic Service-oriented Architecture for e-Government," *Business Services: Konzepte, Technologien, Anwendungen: 9. Internationale Tagung Wirtschaftsinformatik 2009*, vol. 246, February 2009.

[7] K. Lawrence, C. Kaler, A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker, "WS-Security Core Specification 1.1," *OASIS Standard Specification*, 2006.

[8] OASIS, "Web Services Security UsernameToken Profile 1.1," *OASIS Standard Specification*, 2006.

[9] OASIS, "Web Services Security X.509 Certificate Token Profile 1.1," *OASIS Standard Specification*, 2006.

[10] OASIS, "Web Services Security: SAML Token Profile 1.1," *OASIS Standard*, 2006.

[11] J. Hughes, E. Maler, and R. Philpott, "Technical Overview of the OASIS Security Assertion Markup Language (SAML) V1.1," *OASIS Standard Specification, Committee Draft*, 2004.

[12] T. Moses, "eXtensible Access Control Markup Language (XACML) Version 2.0," *OASIS Standard*, 2005.

[13] A. Anderson, H. Lockhart, and B. Parducci, "Web Services Profile of XACML (WS-XACML) Version 1.0," *OASIS Standard Specification, Working Draft 8*, 2006.

[14] A. Anderson and H. Lockhart, "SAML 2.0 profile of XACML v2.0," *OASIS Standard Specification*, 2005.

[15] S. Cantor, F. Hirsch, J. Kemp, R. Philpott, and E. Maler, "Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0," *OASIS Standard Specification*, 2005.

[16] WSMO Project, "D2v1.2. Web Service Modeling Ontology (WSMO)," tech. rep., 2005.

[17] J. de Bruijn, H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu, M. Kifer, and D. Fensel, "The web service modeling language WSML," *WSML Final Draft D*, 2005.

[18] P. Patel-Schneider, P. Hayes, I. Horrocks, *et al.*, "OWL web ontology language semantics and abstract syntax," *W3C recommendation*, vol. 10, 2004.

[19] J. Kopecký and D. Roman, "WSMOgrounding," *WSMO Working Draft*, 2007.

[20] J. Farell and H. Lausen, "Semantic Annotations for WSDL (SAWSDL)–W3C Working Draft 28 September 2006,"

[21] M. Skokan and P. Bednar, "Orchestration of public administration services with a use of semantic technologies," *The 6th Slovakian - International Symposium on Applied Machine Intelligence*, 2008.

[22] C. Ardagna, E. Damiani, M. Cremonini, S. De Capitani di Vimercati, and P. Samarati, "The Architecture of a Privacy-aware Access Control Decision Component," *Proc. of the Construction and Analysis of Safe, Secure and Interoperable Smart devices (CASSIS'05)*, 2005.

[23] E. Damiani, S. De Capitani di Vimercati, C. Fugazza, and P. Samarati, "Extending Context Descriptions in Semantics-Aware Access Control," *Proc. of the 2nd International Conference on Information Systems Security (ICISS 2006)*, December 2006.

[24] C. A. Ardagna, M. Cremonini, S. De Capitani di Vimercati, and P. Samarati, "A privacy-aware Access Control System," *Proc. of the 20th Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec'06)*, vol. 16, pp. 369–397, 2008.

[25] Access-eGov, "Access-eGov Security Architecture & Privacy Concept," tech. rep., Access-eGov, 2007.

[26] Access-eGov, "D2.2, User Requirements Analysis & Development/Test Recommendation," tech. rep., Access-eGov, 2006.

[27] J. Lopez, R. Oppliger, and G. Pernul, "Authentication and Authorization Infrastructures (AAIs): A Comparative Survey," *Computers & Security*, vol. 23, pp. 578–590, 2004.

[28] W. Hommel, "Using XACML for Privacy Control in SAML-Based Identity Federations," *IFIP International Federation for Information Processing CMS, LNCS 3677*, pp. 160–169, 2005.