2011 IEEE World Congress on Services

Semantic Security Policy Matching in service oriented architectures

Giuseppe Di Modica, Orazio Tomarchio
Dipartimento di Ingegneria Elettrica, Elettronica e Informatica
Università di Catania
Viale A. Doria 6, 95125 Catania - Italy
Email: {Giuseppe.DiModica, Orazio.Tomarchio}@dieei.unict.it

***Abstract*—Cloud computing poses several new security and privacy challenges, mainly related to resource sharing, interoperability and dinamicity among different providers. Although policy specification languages address some of these challenges, many issues still have to be faced with. Policy matching is today performed by way of syntactical approaches, which may limit the selection of suitable services on the one hand, and the flexibility and the dinamicity of the matching process on the other one. In this work we propose a semantic approach that, by means of semantic annotations to WS-Policy documents, allows for an improved matching of security requirements and capabilities based on their actual meaning. The proposed approach has been validated through a case study that shows how a pure syntactic-based mechanism of WS-Policy would have failed in matching two actually compatible policies.**

***Keywords*-Security policy, Semantic matching, WS-Policy, Cloud computing**

I. Introduction

Security in resource management and access within cloud environment is a key factor enabling for the development and the success of such technology [1]. Today's techniques for guaranteeing adequate levels of confidentiality, integrity and authentication can be efficiently adopted by a Cloud provider [2]. Yet, when considering the interaction among different Cloud providers, there is a clear need to establish strong trust relationships on which to build an environment of shared resources, especially with partners unknown beforehand [3]. Furthermore, the problem of interoperability among the different security systems and mechanisms used by each Cloud provider must be faced. Indeed, cloud computing environments can be seen as multi-domain environments where each domain can use different security, privacy, and trust requirements, potentially leveraging on different mechanisms and protocols.

Today, Service Oriented Architecture [4] is the most relevant technology that facilitates such multi domain cooperation through service discovery, composition and orchestration. WS-Security [5] defines an abstract model for security in Web services, not only addressing messages confidentiality and integrity but also providing models for exchanging various security token and defining access control mechanisms [2]. But, if with respect to the message security there are several well-established techniques and mechanisms, the discovery and compatibility of security requirements among "interoperable services" still lacks of an established methodology. In order to ensure security in the several possible scenarios that may arise, mechanisms must be devised to dynamically verify the compatibility between the consumer's security requirements and the service provider's security capabilities.

The approach mainly adopted today leverages on metadata and languages for the specification of security policies: this permits to abstract from the low-level security mechanisms of the specific cloud infrastructure. The use of languages for the specification of security policies also enables, on the one hand, the providers to expose the security capabilities implemented in their administrative domains and, on the other one, the users to express their security requirements.

In the context of WS-Security, WS-Policy [6] is the reference language to express policies for Web services. It is used to express requirements, preferences and capabilities of services, allowing for both simple declarative assertions as well as more sophisticated conditional assertions. However, the comparison of policies within the WS-Policy framework is done at a syntactic level, thus limiting the selection of suitable Web services.

As a simple example let us consider a scenario where a policy assertion states that the service requestor requires data encryption through DES algorithm, while the service provider just supports the XML-Enc encryption standard. In this scenario, if we were to use the WS-Policy model for matching the policies, we would get a no-match (a false negative) because no additional knowledge is available on both the "DES" algorithm and the "XML-Enc" standard.

Our approach aims to add a semantic knowledge to policy assertions, by defining a common security ontology, allowing for a semantic matching between security requirement and capabilities. Following our approach, in the above example we are able model the fact that XML-Enc is an encryption standard that also supports the DES algorithm: this additional knowledge about the two assertions will lead us to a positive match (which is actually the correct result). Our approach does not propose a new semantic policy language, unlike other authors do [7], [8], but leverages on the WS-Policy, allowing for a lightweight approach where existing syntactic policies could be processed by our semantic-aware tools and, viceversa, semantic policies could

978-0-7695-4461-8/11 $26.00 © 2011 IEEE
DOI 10.1109/SERVICES.2011.110

coexist within WS-Policy frameworks (although they can not be semantically processed).

The rest of the paper is structured as follows. Section 2 presents background and related work. Then in Section 3 the overall architecture of our framework is presented, together with the description of the security ontology and the matching algorithm. In Section 4 an example is reported to validate our approach. Finally we conclude the work in Section 5.

II. Related work

Policy-based management has been widely employed in enterprise information system [9], where policies are often applied to automate network administration tasks. Multiple approaches for policy specification have been proposed that range from formal policy languages that can be directly processed by a computer, to rule-based policy notations, and to representation of policies as entries in a table consisting of multiple attributes. Within the last few years policy management has extended its original scope, going beyond traditional domains.

In the Web service domain policies are used to express non-functional service properties: WS-Policy [6] is the specification used in this context. A policy is defined as a collection of alternatives and each alternative is a collection of assertions. Assertions specify characteristics that may be used for service selection such as requirements, capabilities or behaviours of a Web service. Policy assertions can represent both requirements on service requestors and the capabilities of the service itself. Requirements represent a demand on service requestors to follow a particular behaviour; capabilities are the service providers promises of behaviour. Among the others non functional properties of a service, WS-Policy may be used to express security requirements and security capabilities. For example, the use of a specific protocol to protect message integrity is a requirement that a service can impose on requestors. On the other hand, the adoption of a particular privacy policy when manipulating requestors' data is a service capability.

Policy matching in WS-Policy works at a syntactic level: it offers a domain-independent mechanism to find alternatives that are compatible to two policies. Two alternatives are compatible if, for each assertion in an alternative, there is a compatible assertion in the other one. Compatibility of assertions is defined exclusively according to the equality of their *qnames*, without any further mechanism involving their structure and content.

For this reason, several works in the literature [10], [11], [12], [13] have been trying to enhance WS-Policy with semantic annotations. In [10], WS-Policy assertions refer to policies expressed in OWL: however that work is not focused on policy matching, but on modeling policies as a set of rules, which have to be evaluated using an external rule-based system, requiring reasoning beyond OWL. In [11] policies represented in WS-Policy are enhanced with semantics by using a combination of OWL and SWRL based rules to capture domain concepts and rules. In [12] a lightweight approach to specify semantic annotations in WS-Policy is presented: it combines the syntactic matching with the semantic matching capability provided by OWL.

Our work, as the last two ones, carefully extends WS-Policy maintaining backward compatibility with existing tools: we believe this is a strong point when dealing with enterprise and cloud systems that adopt a service oriented paradigm.

However, adopting WS-Policy as base language to express policies may limit the flexibility and the expressiveness of the policy itself; for this reason, several other approaches present ontology-based policy languages that are not based on WS-Policy [14], and are not specifically focused on the security domain. Among the most notably efforts in this domain it is worth citing Kaos [7], a policy management framework where concepts and policies themselves are represented using OWL, and Rei [8], a policy framework where OWL is extended with the expression of relations like role-value maps, making the language more expressive than the original OWL.

III. System Architecture

In this section we discuss the overall architecture of the semantic security policy matching framework. The basic logical view of the architecture, depicted in Figure 1, mainly follows that of classic SOA: the requestor, the provider and the registry keep playing their original role in the publish-find-bind cycle. The novelty is the presence of the *Matchmaker* and the *Reasoner* entities, and that of policy-related information flows.

When a service provider advertises a service, besides providing the service "functional" information in the WSDL document, it also attaches the policy associated with that service. The WS-PolicyAttachment specification defines how to integrate the policy information. This information can be integrated either as an attachment to WSDL description, or as an additional document to the UDDI registry. When a requestor queries the UDDI registry for a service with given functional features, the registry also responds with the service's attached policy. When selecting the service(s) matching the functional requirements, the requestor may decide to evaluate how well the providers' security capabilities specified within the retrieved services' policies match those specified by its own policy.

The Matchmaker is the component that the requestor relies on to perform such an evaluation. It is modelled as a web service, and receives requests with two inputs, i.e., the policies specifying the requirements and the capabilities of, respectively, the service requestor and the service provider. The specification of both policies is done by extending WS-Policy.

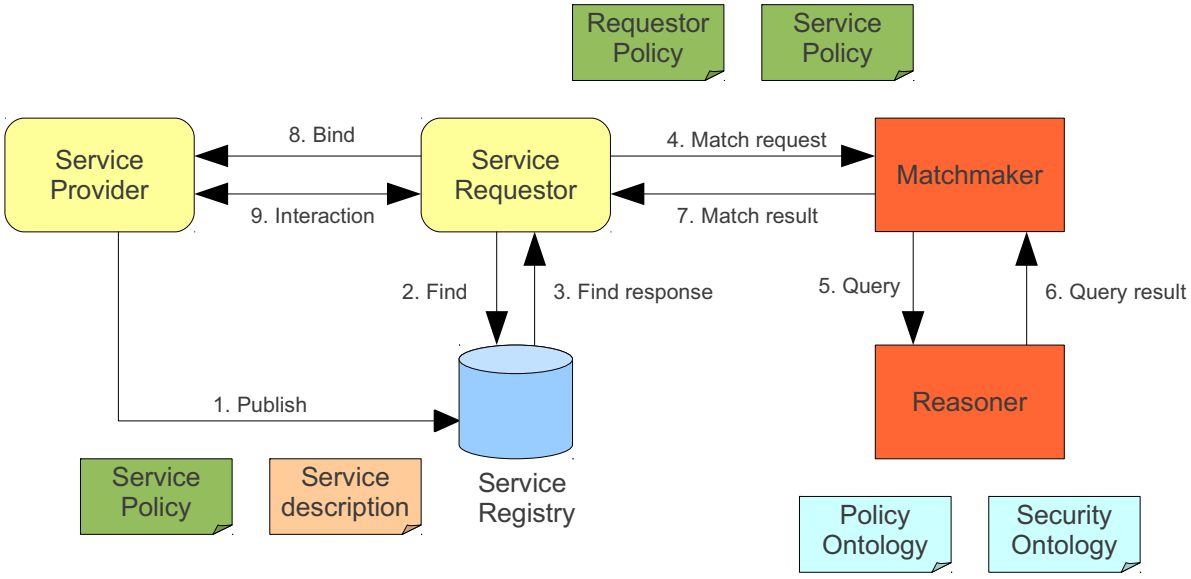

Figure 1. Overall architecture

Since the specification does not impose limits on the kind of allowed assertions, as already said, for the definition of the policies we have decided to adopt semantically enriched terms. This, of course, will enable semantic-based procedures of policy comparisons.

The requestor's and the provider's policies are confronted by overlapping the requirements defined in one policy on the capabilities expressed in the other, and viceversa. This way every possible requirement-capability pair is assigned a match level and, in the end, the overall match level between the requestor and the provider is obtained. Since requirements and capabilities are expressed in a semantic form, the Matchmaker will invoke the Reasoner component to perform the semantic match. To accomplish its task, the Reasoner makes use of the *Security Ontology*, which is an ontology that describes concepts from the security domain, expressly created to let the Reasoner evaluate the relationships among the semantic concepts that populate the policies.

A. Security and Policy ontology

To support the semantic description of security requirements and capabilities two ontologies have been defined: a *security ontology*, that describes security related concepts like protocol, algorithm, credential, and a *policy ontology* that defines the concept of the policy and its characterization in term of requirements and capabilities. In the following more details about the ontologies are provided. We point out that it is not among the objectives of this work to provide an exhaustive view of all concepts that populate the domain of security.

In the literature many have tried to define ontologies for security. The ontology proposed in [15], that makes explicit references to the WS-Security's nomenclature, addresses the problem of security when messages are exchanged among web services. In [16] the proposed ontology covers most of the concepts of the security domain: despite it was defined to address security aspects at a very high levels, there are some constructs expressly designed to semantically annotate web services.

The security ontology we propose, depicted in Figure 2, takes inspiration from the solutions proposed in literature. The main concepts at the base of the security domain are *Protocol*, *Algorithm*, *Credential* and *Objective*.

A *Protocol* is a set of rules agreed by parties that need to communicate to each other. In the context of security a protocol makes use of tools, like algorithms and credentials, in order to accomplish an objective. An *Algorithm* is a procedure for solving problems that employs a finite number of steps. In the literature several security algorithms, divided in as many categories, have been proposed. We can cite, for instance, the category of encryption and that of authentication algorithms. As for the former, another categorization can be proposed according to the type of key used for the encryption: so we distinguish between symmetric (e.g. DES, 3DES, AES) and asymmetric algorithms (e.g. RSA). Among the authentication algorithms, we identify those that make use of hash functions (e.g. SHA, MD5) and those that make use of MAC functions (e.g. HMAC, CBC-MAC). The protocol classification resembles that of algorithms. We can have encryption protocols (e.g. XML-Enc), signature protocol (e.g. XML-Dsig), authentication protocols (e.g. SAML), protocols for access control (e.g. XACML), protocols for key management (e.g. XKMS) and

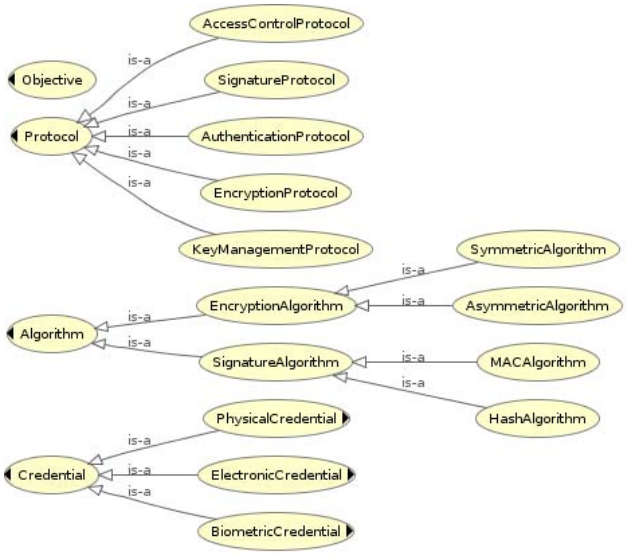

Figure 2. Security ontology

authorization protocols. *Credentials* play an important role in information systems that require authentication. Again, there are several categories of credentials, among which we can cite the biometric (fingerprint, voice), electronic (login, password, encrypted keys, certificates), physical (smartcard, passport, identity card). Finally, by *Objective* we mean a particular security service offered by the system. The following services can be offered: authentication, authorization, confidentiality, integrity and non-repudiation.

The concepts are related to each other within the ontology, and some properties (not shown in figure for the sake of clarity) have been also defined:

- a security protocol makes use of one or more security algorithms (*hasAlgorithm* property);
- a protocol requires one or more credentials (*reqCredential* property);
- a protocol supports one or more security objective (*hasObjective* property).

The OWL language has been used to formalize the just described ontology [17].

The policy ontology is a very short ontology that defines the concept of policy in the context of our architecture. A policy is nothing but a list of requirements and capabilities. In the OWL formalization, the Policy Class and the related properties *hasRequirement* and *hasCapabilities* have been created.

We give an example of policy defined through the concepts described in the security and policy ontologies:

```
<security:Authentication rdf:ID="requirement0"/>
<security:XMLEnc rdf:ID="requirement1"/>
<security:XMLDsig rdf:ID="capability0">
    <security:hasSignatureAlgorithm
        rdf:resource="&security;MD5"/>
</security:XMLDsig>

<policy:Policy rdf:ID="policy0>
  <policy:hasRequirement
      rdf:resource="#requirement0"/>
  <policy:hasRequirement
      rdf:resource="#requirement1"/>
  <policy:hasCapability
      rdf:resource="#capability0"/>
</policy:Policy>
```

This policy is composed of two requirements and one capability. The first requirement imposes the authentication objective. The second requires the use of the XML-Enc protocol. In the policy, also, the signature protocol XML-Dsig is provided, and for this protocol the use of the MD5 hashing algorithm is specified. A policy expressed in WS-Policy is translated into the above described format through an XSLT transformation. This format eases the task of the Matchmaker's matching algorithm.

B. Matching algorithm

In this section we provide some details on how the matching algorithm works. The process of making the match between two policies consists in searching a correspondence between requirements and capabilities. Specifically, a) the requirements of a service are compared to the capabilities of the requestor, and b) the capabilities of the service are compared to the requirements of the requestor. In order for the comparison process to have a positive outcome, the two following conditions must hold:

- the capabilities expressed in the service's policy must meet the requirements expressed in the requestor's policy;
- the requirements expressed in the service's policy must be met by the capabilities expressed in the requestor's policy.

The matchmaking process breaks down in two steps: assigning the match level to each requirement-capability pair and assigning the match level between to overall set of requirements and the overall set of capabilities.

As for the first step, each requirement-capability pair can be assigned one out of a scale of four different match levels:

- Perfect Match;
- Close Match;
- Possible Match;
- No Match;

For each requirement, the objective is to find the capability that matches at best. In the second step the overall match between the two policies will be evaluated. The overall match is defined to be the minimum among the individual match levels evaluated in the first step for each requirement-capability pair.

In the following a detailed description of the four levels of match is given.

Perfect Match: A perfect match occurs when the requirement and the capability both refer to the same concept, or to two equivalent concepts. In particular, two cases are possible: a) the requirement and the capability refer to the very same semantic concept. If properties are also specified, their values are identical; b) the requirement and the capability refer to equivalent semantic concepts and, if specified, the properties are identical or equivalent.

Close Match: A close match occurs when the requirement is more general then the capability. Three cases are possible: a) the requirement specifies a more general semantic concept than the capability's, i.e., a concept that is higher in the ontological hierarchy; b) the requirement and the capability refer to the same semantic concept, but more details are specified for the capability (using the "property" construct); c) referring to the security ontology, the requirement is expressed in terms of security objective while the capability is expressed in terms of security concept that supports the specified objective.

Possible Match: A possible match is when the requirement is more specific than the capability. It can be presented as the opposite of the Close Match condition. Three cases are possible: a) the requirement specifies a more specific semantic concept than the capability's, i.e., lower in the ontological hierarchy. b) the requirement and the capability refer to the same semantic concept, but the requirement is specified more in detail (using the property construct); c) with reference to the security ontology, the requirement is expressed as security concept while the capability is expressed in term of security objective supported by that concept. When the overall matching result gives a possible match level, a further negotiation step is needed to effectively verify the compliance with the requirement.

No Match: A no match occurs when the requirement and the capability have no chance to match. Two cases are possible: a) the requirement and the capability refer to semantic concepts that have no semantic relationship; b) the requirement and the capability refer to the same semantic concept but have a different specifications for their properties

IV. Case study

In this section, an example case study of the matching algorithm within the semantic security policy framework previously described is presented.

The overall scenario is the classic one (see also Figure 1) in which a client (service requestor) is looking for a service that provides a specific functionality (expressed through the WSDL specification); in addition, the client expresses also security requirements (within its own security policy) that the service provider must meet. The client performs this search on the UDDI registry in order to obtain service information. The registry, besides providing the information necessary to invoke the service (the WSDL specification describing it), also returns the associated security policy.

At this point, our specific components (Matchmaker and Reasoner) come into play, allowing the client to verify the fulfilment of its security requirements. The Matchmaker, once received the two policies, applies the matching algorithm described in the previous section and, with the help of the Reasoner, evaluates their match level. Finally the client, based on this value, decides whether to invoke the service or not.

In the following we will focus on the security policies and their meaning, and on the evaluation of the matching algorithm, also explaining why a standard WS-Policy would have instead failed.

A. Security policies definition

The definition of security policies for both the client requirements and the server capabilities essentially requires the use of two tools: WS-Policy, and the security ontology. The first serves as a container for the requirements and capabilities expressed through the concepts defined within the ontology.

Client policy: The service requestor expresses its policy in terms of requirements and security capabilities. The former are the security requirements imposed to service providers, while the latter express the security capabilities that the client is able to offer. In our example the client defines the followings:

- *Client security requirements*
  - XML-Dsig protocol with SHA-1 for message signature
- *Client security capabilities*
  - SAML protocol with X.509 certificate as authentication credential
  - XML-Enc for message encryption

This policy is expressed in our system in the following way:

```
<?xml version="1.0"?>

<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/
                                    ws/2004/09/policy"
 xmlns:rdf="http://www.w3.org/
                        1999/02/22-rdf-syntax-ns#"
 xmlns:security="http://www.semanticweb.org/
                        ontologies/security.owl#">

<wsp:ExactlyOne>
<wsp:All>

<security:XML-Dsig rdf:ID="requirement0">
   <security:hasSignatureAlgorithm
      rdf:resource="http://www.semanticweb.org/
                        ontologies/security.owl#SHA-1"/>
</security:XML-Dsig>
<security:X.509Certificate rdf:ID="X.509"/>
<security:SAML rdf:ID="capability0">
   <security:reqCredential rdf:resource="#X.509"/>
</security:SAML>
<security:XML-Enc rdf:ID="capability1"/>

</wsp:All>
</wsp:ExactlyOne>

</wsp:Policy>
```

Service policy: The service provider expresses its security policy by defining requirements and capabilities too. In our example the service is supposed to define the followings:

- *Service security requirements*
  - client authentication
  - XML-Enc protocol with DES algorithm for message encryption
- *Service security capabilities*
  - message integrity

This policy is expressed in our system in the following way:

```
<?xml version="1.0"?>

<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/
                                    ws/2004/09/policy"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:security="http://www.semanticweb.org/
                               ontologies/security.owl#">

<wsp:ExactlyOne>
<wsp:All>
<security:Authentication rdf:ID="requirement0"/>
<security:XML-Enc rdf:ID="requirement1"/>
   <security:hasEncryptionAlgorithm
      rdf:resource="http://www.semanticweb.org/
                          ontologies/security.owl#DES"/>
</security:XMLEnc>

<security:DataIntegrity rdf:ID="capability0"/>

</wsp:All>
</wsp:ExactlyOne>

</wsp:Policy>
```

B. Matching algorithm results

The matching algorithm tries to combine the client requirements with the server capabilities (and viceversa) by following the approach we are going to describe:

- The client's requirement of an XML-Dsig protocol with SHA-1 signature algorithm is combined with the service's capability to provide data integrity; the resulting level of match is *Possible Match* because the XML-Dsig protocol supports the objective of data integrity. In this case, a further investigation by the client is needed in order to determine how the service supports the data integrity feature.
- The service requirement related to client authentication is satisfied by the client capability to use SAML with X.509 certificate for authentication purposes. The resulting match level is Close Match, because the objective of authentication is satisfied by the SAML protocol.
- The service requirement of using XML-Enc for encryption is satisfied by the client using the same protocol for this purpose. The resulting match level is Possible Match, because both the requirement and the capability express the same concept, but the requirement is defined with greater detail.

Client requirements	Service capabilities	Match level
XML-Dsig with SHA-1 signature algorithm	Data integrity	Possible match

Table I
CLIENT REQUIREMENTS AND SERVICE CAPABILITIES MATCHING

Service requirements	Client capabilities	Match level
Client authentication	SAML with X.509 certificate	Close match
XML-Enc with DES algorithm	XML-Enc	Possible match

Table II
SERVICE REQUIREMENTS AND CLIENT CAPABILITIES MATCHING

The results are summarized in tables I and II, that respectively show the match of client requirements vs service capabilities and the match of service requirements vs client capabilities; it may be noticed that the lowest level in all pairs formed by the three capability-requirement combinations is *Possible Match*.

The Matchmaker communicates to the client that the selected service can satisfy its security requirements. In this case, the client will have to require additional information or negotiate with the service provider before actual invoking the service (this last step is outside the scope of this work).

We would like to outline that, if using the standard WS-Policy model, the above policies would not have matched since there is no explicit knowledge 1) that Data integrity could be supported by XML-Dsig and 2) that client authentication could be performed by SAML mechanisms.

V. CONCLUSION AND FUTURE WORK

Security has been acknowledged as being one of the primary concern preventing a wide cloud computing adoption. Among the various security issues, the definition, the modeling and the exactly matching of service security policies is a crucial problem that needs to be faced, especially when dealing with service and cloud providers spanning multiple domains. The existing models based on syntactic approaches like WS-Policy are not very well suited for these heterogeneous and dynamic scenarios.

In this paper we proposed to leverage the existing WS-Policy model with semantic annotations, which provides for semantic matching capabilities, allowing to go beyond the strict syntactic intersection of policy assertions. A security ontology allows to catch the relationships among the concepts of security Objective, Protocols, Algorithms and Credentials. A simple example has also been proposed to show the viability of the approach and the actual limits of the pure syntactic-based approaches.

Future works will be aimed to improve the capability to express more complex policies and to enhance the inference ability of the semantic tools.

ACKNOWLEDGEMENTS

The work described in this paper has been partially supported by the MIUR-PRIN 2008 project *"Cloud@Home: a New Enhanced Computing Paradigm"* [18], [19].

REFERENCES

[1] H. Takabi, J. B. Joshi, and G.-J. Ahn, "Security and Privacy Challenges in Cloud Computing Environments," *Security & Privacy, IEEE*, vol. 8, no. 6, pp. 24–31, 2010.

[2] S. Lakshminarayanan, "Interoperable security standards for web services," *IT Professional*, vol. 12, no. 5, pp. 42 –47, Sep/Oct 2010.

[3] G. Peterson, "Don't Trust. And Verify: A Security Architecture Stack for the Cloud," *IEEE Security & Privacy Magazine*, vol. 8, no. 5, pp. 83–86, Sep. 2010.

[4] M. P. Papazoglou and W.-J. van den Heuvel, "Service Oriented Architectures: approaches, technologies and research issues," *VLDB Journal*, vol. 16, no. 3, pp. 389–415, Jul. 2007.

[5] OASIS, "Web Services Security (WS-Security)," OASIS Standard, feb 2006. [Online]. Available: http://www.oasis-open.org/committees/wss/

[6] W3C, "Web services policy 1.5 - framework," W3C Recommendation, sep 2007. [Online]. Available: http://www.w3.org/TR/ws-policy/

[7] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott, "Kaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement," in *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, ser. POLICY '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 93–.

[8] L. Kagal, T. Finin, and A. Joshi, "A policy language for a pervasive computing environment," in *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, ser. POLICY '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 63–.

[9] T. Phan, J. Han, J. Schneider, T. Ebringer, and T. Rogers, "A survey of policy-based management approaches for Service Oriented Systems," in *Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on*. IEEE, 2008, pp. 392–401.

[10] N. Sriharee, T. Senivongse, K. Verma, and A. Sheth, "On using ws-policy, ontology, and rule reasoning to discover web services," in *Intelligence in Communication Systems*, no. May 2004. Springer, 2004, pp. 246–255.

[11] K. Verma, R. Akkiraju, and R. Goodwin, "Semantic matching of Web service policies," in *Semantic Web Policy Workshop (SDWP 2005)*, 2005.

[12] S. Speiser, "Semantic Annotations for WS-Policy," in *IEEE International Conference on Web Services (ICWS 2010)*. IEEE, 2010, pp. 449–456.

[13] H. Zheng-qiu, W. Li-fa, H. Zheng, and L. Hai-guang, "Semantic Security Policy for Web Service," in *2009 IEEE International Symposium on Parallel and Distributed Processing with Applications*. Ieee, 2009, pp. 258–262.

[14] G. Tonti, J. Bradshaw, R. Jeffers, R. Montanari, N. Suri, and A. Uszok, "Semantic Web languages for policy representation and reasoning: A comparison of KAoS, Rei, and Ponder," in *International Semantic Web Conference (ISWC2003)*. Florida (USA): Springer, 2003, pp. 419–437.

[15] D. Z. G. a. Garcia and M. B. Felgar de Toledo, "Ontology-Based Security Policies for Supporting the Management of Web Service Business Processes," in *2008 IEEE International Conference on Semantic Computing*. Ieee, Aug. 2008, pp. 331–338.

[16] A. Kim, J. Luo, and M. Kang, "Security ontology for annotating resources," in *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*. Springer, 2005, pp. 1483–1499.

[17] W3C, "OWL 2 Web Ontology Language," W3C Recommendation, 2009. [Online]. Available: http://www.w3.org/TR/owl2-overview/

[18] "Cloud@Home Project," 2010, https://cloudathome.unime.it/.

[19] R. Aversa, M. Avvenuti, A. Cuomo, B. Di Martino, G. Di Modica, S. Distefano, A. Puliafito, M. Rak, O. Tomarchio, A. Vecchio, S. Venticinque, and U. Villano, "The Cloud@Home project: Towards a new enhanced computing paradigm," in *Workshop on Cloud Computing: Project and Initiatives (in conjunction with EUROPAR2010)*. BERLIN – DEU: Springer Verlag, Aug 31 - Sep 3 2010, pp. –.