Future Generation Computer Systems 55 (2016) 176–185

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Matchmaking semantic security policies in heterogeneous clouds

Giuseppe Di Modica, Orazio Tomarchio *

Department of Electrical, Electronic and Computer Engineering, University of Catania, Viale A. Doria, 6 - 95125 Catania, Italy

HIGHLIGHTS

- Semantic matchmaking of security policies in cloud environments.
- Security ontology for modeling security concepts.
- Automatic semantic annotation of WS-SecurityPolicy policies.

ARTICLE INFO

Article history:
Received 30 June 2014
Received in revised form
11 January 2015
Accepted 9 March 2015
Available online 23 March 2015

Keywords:
Cloud computing
Security policies
Semantic
Ontology

ABSTRACT

The adoption of the cloud paradigm to access IT resources and services has posed many security issues which need to be cared of. Security becomes even a much bigger concern when services built on top of many commercial clouds have to interoperate. Among others, the value of the service delivered to end customers is strongly affected by the security of network which providers are able to build in typical SOA contexts. Currently, every provider advertises its own security strategy by means of proprietary policies, which are sometimes ambiguous and very often address the security problem from a non-uniform perspective. Even policies expressed in standardized languages do not appear to fit a dynamic scenario like the SOA's, where services need to be sought and composed on the fly in a way that is compatible with the end-to-end security requirements. We then propose an approach that leverages on the semantic technology to enrich standardized security policies with an ad-hoc content. The semantic annotation of policies enables machine reasoning which is then used for both the discovery and the composition of security-enabled services. In the presented approach the semantic enrichment of policies is enforced by an automatic procedure. We further developed a semantic framework capable of matchmaking in a smart way security capabilities of providers and security requirements of customers, and tested it on a use case scenario.

1. Introduction

Cloud computing technology has definitely reached the commercialization phase of development. Several commercial cloud providers offer a variety of services, ranging from the IaaS to the SaaS, which users can access to build up cloud services for disparate purposes. However, if on the one hand the provision of computing resources as if they were a utility (such as the electricity) is potentially revolutionary as a computing service, on the other one presents many major problems on information policy, including privacy, security, reliability and access control issues [1].

High-level security constraints about the usage and the access to cloud services and resources should be logically separated from the service itself, and expressed in such a way to be interoperable among the different cloud providers' low level security mechanisms. This would enable secure scenarios of world-wide and cross-domain interoperability among services, in line with the perspective of a services *ecosystem* fostered by the SOA paradigm. One of the approaches adopted to get a secure environment of interoperable cloud services relies on cloud federations. In a cloud federation an individual customer is granted the same level of security, no matter the specific cloud system he decides to access. Though profitable to the customer, this approach requires that the participating cloud systems reach a common mutual trust, therefore the end-user's security will be affected by the trust level eventually established among providers. If cloud federations have reached consensus in academic contexts, there is skepticism that the business model beneath it will appeal commercial players. The approach we support, and that is source of inspiration of this work, is instead to put the end-user in the condition of building up a custom security environment, i.e., an environment which is compatible with their

* Corresponding author.
E-mail addresses: Giuseppe.DiModica@dieei.unict.it (G. Di Modica), Orazio.Tomarchio@dieei.unict.it (O. Tomarchio).

http://dx.doi.org/10.1016/j.future.2015.03.008
0167-739X/

specific security requirements. The basic idea is to put in the end-user's hand a tool to discover which of the available and heterogeneous cloud services match both functional and security criteria; with those services the end-user will be able to build up their secure service context.

Today, the main approach followed to express high-level security constraints is based on the usage of metadata and languages for the specification of security policies [2]. It allows to abstract from the low-level security mechanisms of the specific provider's infrastructure. The benefit of using languages for the specification of security policies is twofold. On the one hand the providers are able to advertise the security capabilities implemented within their administrative domain; on the other one the customers can easily express the security requirements they need for their applications.

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 all the 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 we propose calls on well-known policy and security specifications. We define customers' and providers' security requirements/capabilities within policies which are compliant to well-established specifications, such as WS-SecurityPolicy [3]. As it will be clear further in the paper, one drawback of WS-SecurityPolicy is that it only provides for syntactic matching of security policies. In fact, security policy matching depends on the policy intersection mechanism provided by WS-Policy [4]. What characterizes our approach is that security requirements and capabilities are *semantically* enriched, thus enabling the employment of smarter mechanisms to make the match between what is required and offered in terms of security, respectively on the customer and on the provider side. The contribution of this work, which extends that presented in [5], consists in the design of an ontology defining main security concepts, a procedure which automatically maps syntactic security requirements into semantically-enriched concepts, and a matchmaker engine which is capable of reasoning on the customer's and the provider's security requirements/capabilities to derive a match level.

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. Section 4 presents the semantic model we designed. In particular, we thoroughly discuss the developed security ontology, provide details on the mapping procedure and on the policy matching algorithm. In Section 5 two example case studies are reported to validate our approach. Finally we conclude the work in Section 6.

2. Related work

The adoption of a policy based-approach for managing a system requires an appropriate language for policy representation and modeling and the design and development of a policy management framework for policy matching and enforcement. Several policy languages and framework have been developed following different approaches, and sometimes have been proposed in different application domains [6]. Among the most notable efforts in this domain, worth citing are Ponder [7], a declarative object-oriented language that supports the specification of several types of management policies for distributed object systems, Kaos [8], a policy management framework where concepts and policies themselves are represented using OWL, and Rei [9], 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. Kaos and Rei are ontology based policy languages that, although not specifically focused on the security domain, are able to express complex security policies: policy matching is also supported by the advanced reasoning capabilities these languages offer. However, when dealing with enterprises and cloud systems that adopt a service oriented paradigm, we believe that an approach compatible with existing standard for policy specification should be followed as far as possible.

WS-Policy [4] is the specification used in service oriented architectures to express policies. A policy is defined as a collection of alternatives and each alternative is a collection of assertions. Assertions specify characteristics that can be used for service selection such as requirements, capabilities or behaviors of a Web service. Policy assertions can represent both requirements on service requesters and capabilities of the service itself. Requirements represent a demand from service requesters to the service provider to behave in a particular way; capabilities are the service providers promise of behavior. Within the WS-Policy framework, the WS-SecurityPolicy [3] specification can be used to express security requirements and security capabilities in the form of policies. For example, the use of a specific protocol to protect message integrity is a requirement that a service can impose on requesters. On the other hand, the adoption of a particular privacy policy when manipulating data of a requestor 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 one alternative, there is a compatible assertion in the other one. Compatibility of assertions is defined exclusively according to the equivalence of their *qnames*, without any further mechanism involving either their structure or their content.

For this reason, several works in the literature [10–13] 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 described in next sections, carefully extends WS-Policy by referencing concepts from a security ontology directly in the policy assertions, thus maintaining backward compatibility with existing tools.

3. System architecture

In this section we discuss the overall architecture of the semantic security policy matching framework. The basic architecture of the system, depicted in Fig. 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 (depicted in red), and that of the policy-related information flows (steps 4 through 7).

It is a matter of fact that most of cloud services offered by commercial providers are REST-based. Some providers (Amazon, to cite one) still offer a subset of their services through the SOAP interface, thus providing the relevant WSDL service representation. The architecture depicted in Fig. 1 is compatible with both the REST and the SOAP approach, in the sense that it is capable of performing the security match of services no matter how they have been advertised in the Service Registry and independently of which interface they are going to be finally accessed through. The

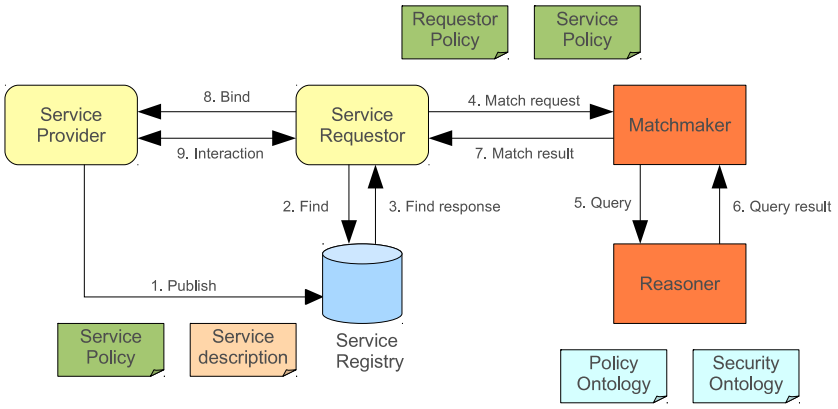

Fig. 1. Overall architecture.

Service description is the document describing the service features. In the case of a SOAP service, this will be a regular WSDL document specifying the service end-point and the operations that may be invoked on it. In the case a service is REST-based, the provider will have to supply either only the service end-point or a full WSDL representation of the service. Since the WSDL specification v2.0 [14], in fact, REST-based services can be easily described in a WSDL document as well as SOAP's. Either way, the matchmaking mechanism (steps 4 through 7) is capable of fulfilling its task, as it only deals with policy-related information. The service matchmaking step may be reasonably considered independent from (though fully integrated to) the rest of the SOA operations. For our purpose, in the rest of the paper we are going to assume that cloud services' representation is available in the WSDL form, no matter whether they are accessible via REST or via SOAP.

Let us suppose that a cloud provider (*Service Provider* in the figure) wants to advertise a service. Besides supplying the service "functional" information in the WSDL document (*Service Description*), it also has to define the policy document (*Service Policy*) to which the service refers to.

The WS-PolicyAttachment specification defines how to integrate the policy information. This information can be integrated either as an attachment to the *Service description*, or by adding it to the *Service Registry* as an extra document. When a requestor queries the registry for a service with given functional features, the registry will also reply with the service's attached policy.

When selecting the service(s) that matches 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 can rely on to perform such an evaluation. It is modeled as a web service, and receives requests with two inputs: 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. Since the specification does not impose limits on the kind of allowed assertions, for the definition of the policies we decided to adopt semantically enriched terms. This, of course, will enable semantic-based procedures of policy comparisons.

In order to cope with existing policy specifications, we developed a module which transforms a standard WS-Policy containing WS-SecurityPolicy assertions into a policy expressed by using semantic concepts belonging to our ontology. This allowed us to integrate existing systems relying on plain WS-Policy into our framework.

The requester's and the provider's policies are compared by overlapping the requirements defined in one policy and the capabilities expressed in the other, and vice versa. This way every possible requirement-capability pair is assigned a match level, and the overall match level between the requestor and the provider is eventually 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, created on purpose to let the *Reasoner* evaluate the relationships among the semantic concepts that populate the policies.

4. Semantic model

In this section the semantic model proposed by our approach is described. An ontology was developed to represent main concepts from the security domain. Such ontology will be used to define concepts of security within the policies. Also, we developed a procedure which is able to automatically map security concepts expressed in the WS-SecurityPolicy specification onto concepts of our security ontology. This procedure enables providers and customers adopting the WS-SecurityPolicy specification to seamlessly use our framework. Finally, to help customers finding the providers that best suite their security requirements, a semantic discovery engine was developed which is capable of reasoning on customers' requests and suggesting highly matching providers.

4.1. 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 terms 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.

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.

For what concerns the security ontology we take inspiration from the solutions proposed in the literature. As an example, 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 that it was defined to address security aspects at

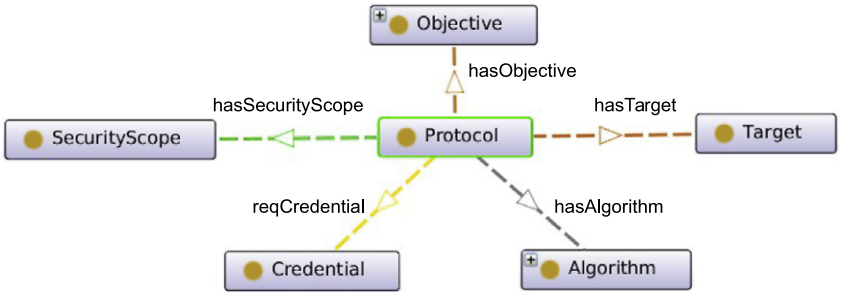

Fig. 2. Security ontology: top level classes.

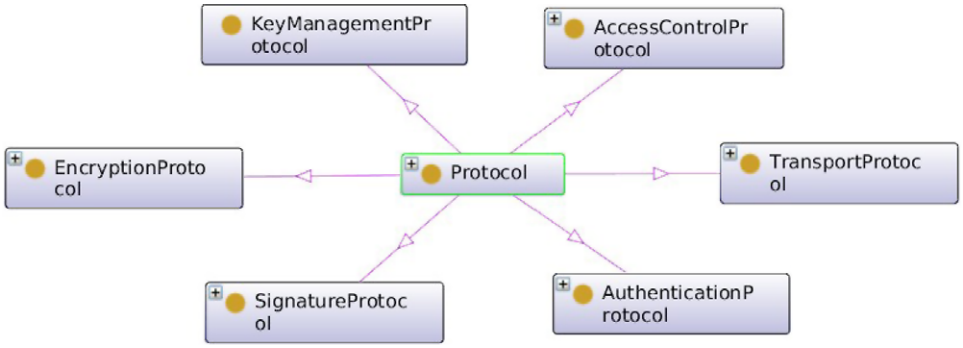

Fig. 3. The protocol class.

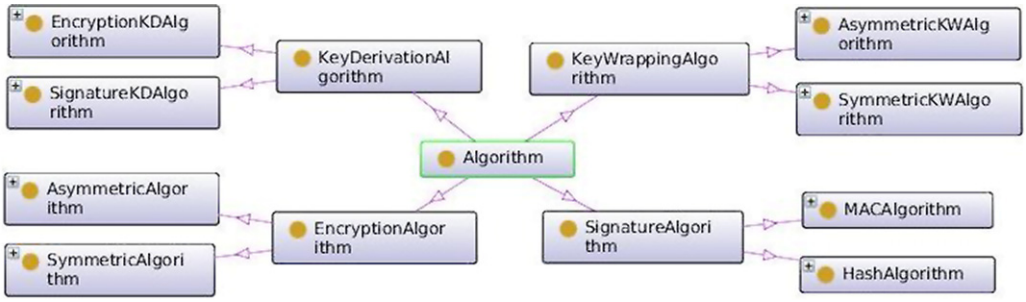

Fig. 4. The algorithm class.

a very high levels, there are some constructs expressly designed to semantically annotate web services. In this work we extended and refined a preliminary ontology [17,18], in order to adequately represent security concepts from the WS-SecurityPolicy specification.

A graphic overview of our proposed security ontology is depicted in Fig. 2: the main concepts are *Objective, Protocol, Algorithm, Credential, Security scope* and *Target*.

A security *Objective* is a high level abstraction by which we mean a particular security service offered by the system. The following services can be offered: authentication, authorization, confidentiality, integrity and non-repudiation.

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. In Fig. 3, the different protocol categories are represented: we may distinguish encryption, signature, transport, authentication, access control and key management protocols. Each of these categories may contain several individuals, reflecting different security protocols such as, for example, XML-Enc, XML-Dsig, SAML, and XACML.

An *Algorithm* is a procedure for solving problems that employs a finite number of steps. In the literature several security algorithms have been proposed. We can cite, for instance, the category of encryption and that of authentication algorithms. For the former, another categorization can be proposed according to the type of key used for the encryption: symmetric (e.g. DES, 3DES, AES) and asymmetric (e.g. RSA). Among the authentication algorithms, instead, we identify those that employ hash functions (e.g. SHA, MD5) and those using MAC functions (e.g. HMAC, CBC-MAC). We also included the categories of Key derivation and Key wrapping algorithms. The resulting security algorithm branch is depicted in Fig. 4.

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), and physical (smartcard, passport, identity card).

The *Security scope* class allows to differentiate between security concepts related to a single host and security in communication (like message, transport or network level security).

Finally, the *Target* class allows to model the target which the specific security mechanism is to be applied to. The *Target* is a very generic concept which needs to be further specified (i.e., subclassed) according to the security context that is going to be addressed. In a former work [5] we provided an example where the *Target* concept was subclassed by *MessageParts*, which represented all of the elements that can be found in a message according to the WS-Security specification. Since the objective of this work is

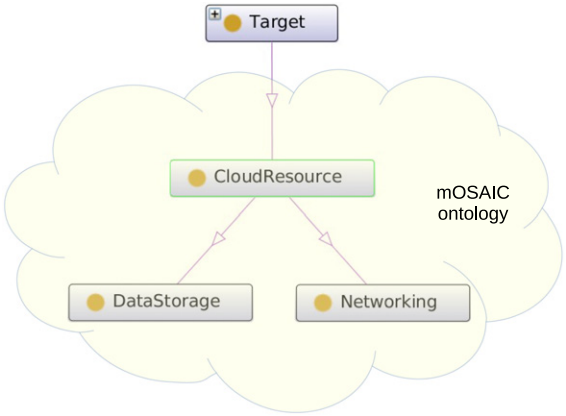

Fig. 5. The target class.

the application of the security framework to the cloud context, we need to define the semantic concepts representing the cloud resources which are going to be targeted by a security mechanism. To this end, we borrowed the *CloudResource* concept from the mOSAIC ontology, which is one authoritative cloud ontology developed by the mOSAIC European project [19], which in its turn is also inspired by the OCCI [20] cloud resource model. In Fig. 5 the relationship between the *Target* and the mOSAIC *CloudResource* concept is represented together with a simple branch of that ontology (the one representing the data storage and the networking resource concepts).

All the concepts shown in Fig. 2 are related to each other by means of object properties. The following relationships among concepts have been defined:

- a protocol supports one or more security objective (*hasObjective* property).
- a protocol makes use of one or more security algorithms (*hasAlgorithm* property);
- a protocol requires one or more credentials (*reqCredential* property);
- a protocol may be applied to a security scope (*hasSecurityScope* property);
- a protocol may protect a specific target (*hasTarget* property).

A policy written according to our ontology has a standard WS-Policy structure: the wsp:Policy, wsp:ExactlyOne and wsp:All tags are present and have the same meaning than the standard specification. The only constraint in order to allow a correct working of our matching algorithm is that the policy should be normalized: it has to contain only one wsp:ExactlyOne element which, in turn, may contain any number of wsp:All elements. Each element contained in an assertion is semantically annotated with concepts of our security ontology: in addition it should be specified if the element represents a requirement or a capability. In the following we give an example of a policy containing only one alternative:

```
<wsp:Policy>
...namespace definition...
  <wsp:ExactlyOne>
    <wsp:All>

      <security:LoginProtocol rdf:ID="requirement0" >
        <security:requiresCredential
           rdf:resource="sec-ns#OneTimePassword"
           security:isMainCredential="false"/>
      </security:LoginProtocol>

      <security:Https rdf:ID="requirement1" >
        <security:hasSecurityScope
         rdf:resource="sec-ns#TransportLevelSecurity"/>
        <security:requiresCredential
           rdf:resource="sec-ns#Timestamp"/>
        <security:hasEncryptionAlgorithm
           rdf:resource="sec-ns#AES-256"/>
        <security:hasSignatureAlgorithm
           rdf:resource="sec-ns#SHA-1"/>
      </security:Https>

    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

The policy file starts with the wsp:Policy element. The next element is wsp:ExactlyOne, which contains just one wsp:All element. The policy is already normalized and contains one assertion. Two requirements are contained in this assertion: security:LoginProtocol, which specifies a login protocol, and security:Https, which specifies a protocol for the communication. The latter is better described by specifying some of the used algorithms, the provided security level and the need for a timestamp. All these properties are specified using concepts defined in the previously described security ontology.

4.2. Policy mapping

We developed a framework module, called **Transformer**, which is capable of transforming a policy, written in WS-Policy and containing assertions specified through the WS-SecurityPolicy, into an equivalent policy whose assertions are semantic concepts defined through the above discussed security ontology.

Let P_{WSSP} be a WS-Policy compliant policy also containing WS-SecurityPolicy assertions. The Transformer implements a procedure that iterates on P_{WSSP}'s alternatives. For each iterated alternative, its semantic equivalent is returned in the case that the alternative is expressed in WS-SecurityPolicy; otherwise, the alternative is returned unchanged. The Transformer is fed with a Map whose keys are the basic WS-SecurityPolicy keywords and values are their equivalent semantic concepts: the Map will be used to semantically enrich the policy document.

At the current stage, not all elements contained in an alternative have a corresponding semantic concept. The reason is twofold. First, the developed ontology focuses just on main security concepts (integration of other concepts is planned in the near future); second, some WS-SecurityPolicy elements deal with technical aspects which are not specifically bound to security (e.g., the layout order). WS-SecurityPolicy terms that have no correspondence in the ontology are simply ignored by the Transformer.

In a preliminary step the Transformer parses P_{WSSP} and creates a tree of nodes representing the WS-SecurityPolicy alternatives. Secondly, the tree is visited to map combinations of alternatives onto new "equivalent" alternatives. Finally, the original WS-SecurityPolicy terms contained in the just created alternatives are replaced by semantic concepts according to the instructions provided by the previously mentioned Map. An excerpt of that Map is reported in Table 1. Due to text formatting constraints, the namespace http://www.semanticweb.org/ontologies/security.owl was replaced by the *sec-ns* prefix.

4.3. Semantic discovery

We implemented a semantic discovery engine capable of searching for correspondences between security requirements and capabilities. Suppose that both the customer (requestor) and the provider (service) have expressed their requirements and capabilities in their own policies. The engine operates this way: (a) the requirements of the service are compared to the capabilities of the requestor; (b) the capabilities of the service are compared to the requirements of the requestor. In order for the comparison

Table 1
Mapping between WS-securitypolcy terms and semantic concepts.

WS-SecurityPolicy term	Semantic concept
Https	sec-ns#Https rdf:ID requirement/capability
XML-Enc	sec-ns#XML-Enc rdf:ID requirement/capability
Login	sec-ns#LoginProtocol rdf:ID requirement/capability
Authentication	sec-ns#Authentication rdf:ID requirement/capability
TransportLevel	sec-ns#hasSecurityScope rdf:resource sec-ns#TransportLevelSecurity
AES256	sec-ns#hasEncryptionAlgorithm rdf:resource sec-ns#AES-256
AES192	sec-ns#hasEncryptionAlgorithm rdf:resource sec-ns#AES-192
AES128	sec-ns#hasEncryptionAlgorithm rdf:resource sec-ns#AES-128
EncryptedBody	sec-ns#encryptsPart rdf:resource sec-ns#Body
SignedBody	sec-ns#signsPart rdf:resource sec-ns#Body
RequiredBody	sec-ns#requiresPart rdf:resource sec-ns#Body
SignSignature	sec-ns#signsPart rdf:resource sec-ns#Signature
SignSignatureToken	sec-ns#signsPart rdf:resource sec-ns#SigningToken
SignSupportingToken	sec-ns#signsPart rdf:resource sec-ns#SupportingToken

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

The first step of the matchmaking process has to assign a match level to each requirement-capability pair. The comparison will leverage on the well-known concepts of *exact*, *subsume*, *plugin*, and *no match* [21], and has been carried out by means of techniques that have already been explored by authors in the context of supply–demand matchmaking in cloud markets [22].

In the second step the overall match between the two policies is evaluated: the objective is to find the capability that matches at best for each requirement. The overall match is then 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 than 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 terms 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.

In Section 5.1.2 a step-by-step example clarifies the way the matching process works in a real scenario.

5. Implementation and case studies

A prototype of the semantic security framework was developed to test the viability of the described approach. Both the security and the policy ontology were designed in the Protege tool. The framework's main entities (Matchmaker, Reasoner) were implemented as Java components. To implement the reasoning mechanisms enforced by the Reasoner component, the OWL APIs and Jena libraries were used.

In the remainder of the section, two simple case studies showing the overall behavior of the semantic security framework are presented. The first one describes a classic SOA scenario where a customer is looking for a service that provides a specific functionality expressed through the WSDL specification, with a specific security policy which has been expressed using WS-SecurityPolicy assertions: it mainly demonstrates how the Transformer component is able to transform a standard policy written in WS-Policy and containing assertions specified through the WS-SecurityPolicy, into an equivalent policy whose assertions are semantic concepts defined through the proposed security ontology. The second example focuses instead on a classic cloud scenario, where a customer is looking for a pool of Vms (virtual cluster) together with some amount of cloud storage. The customer wants to select a provider offering an adequate level of confidentiality for the data storage as well as for the internal communication of VMs of the virtual cluster. Two cloud providers with different security policies are present.

5.1. Case study 1

The overall scenario of this case study regards a classic SOA scenario (see also Fig. 1) where a client (service requestor) is looking for a service that provides a specific functionality (expressed through the WSDL specification), with a specific security policy which has been expressed using WS-SecurityPolicy assertions; in addition, the client expresses also security requirements (within its own security policy) that the service provider must meet. The

client searches the Service Registry in order to retrieve service information. The registry, besides providing the service's functional information, also returns the provider's associated security policy. The Transformer transforms this policy into a semantically annotated policy.

At this point, the Matchmaker and the Reasoner come into play, allowing the client to verify the fulfillment of its security requirements. The Matchmaker is fed with the client's and the provider's policies. It then runs the matching algorithm and, with the help of the Reasoner, evaluates the match level between the policies. Finally the client, based on the obtained value, decides whether to invoke the service or not.

5.1.1. Security policies definition

The service provider expresses its security policy by using the standard WS-Policy framework, adopting WS-SecurityPolicy terms. The specific policy has been taken by the example policies of WS-SecurityPolicy specification [3]. The scenario provides for a requestor that wants to access a service anonymously, and a service provider that presents its credentials. In order to guarantee message confidentiality and authentication, the security protocol will use symmetric ephemeral keys, generated by the provider and exchanged through asymmetric encryption.

Provider policy. The provider policy is composed of multiple items: one at the endpoint level, and two for input and output messages respectively. At the upper level the WS-SecurityPolicy elements are: SymmetricBinding, EncryptedParts, SignedParts and Wss11. Each element, in its turn, is characterized by several properties (not described here for space reason) better detailing the requirements and the capabilities of the provider.

The Transformer module takes as input this policy and provides a corresponding policy annotated with semantic concepts. The resulting policy, which is shown in Listing 1, contains two requirements and three capabilities. In order to improve the readability of the listing, the namespace http://www.semanticweb.org/ontologies/security was removed from each rdf:resource term.

Listing 1: Semantically enriched Policy

```
<wsp:Policy>
 <wsp:ExactlyOne>
 <wsp:All>
  <security:Authentication rdf:ID="capability0"/>
  <security:XML-Enc rdf:ID="capability1">
   <security:hasKeyDerivationAlgorithm rdf:resource=
       "#PSha1L192"/>
   <security:requiresCredential rdf:resource="#
       X_509Certificate"/>
   <security:hasSecurityScope rdf:resource="#
       MessageLevelSecurity"/>
   <security:requiresCredential rdf:resource="#
       Timestamp"/>
   <security:hasEncryptionAlgorithm rdf:resource="#
       AES-256"/>
   <security:encryptsPart rdf:resource="#Body"/>
  </security:XML-Enc>
  <security:XML-Enc rdf:ID="requirement0">
   <security:hasSecurityScope rdf:resource="#
       MessageLevelSecurity"/>
   <security:requiresCredential rdf:resource="#
       Timestamp"/>
   <security:hasEncryptionAlgorithm rdf:resource="#
       AES-256"/>
   <security:encryptsPart rdf:resource="#Body"/>
  </security:XML-Enc>
  <security:XML-Dsig rdf:ID="capability2">
   <security:hasKeyDerivationAlgorithm rdf:resource=
       "#PSha1L256"/>
   <security:requiresCredential rdf:resource="#
       X_509Certificate"/>
   <security:hasSecurityScope rdf:resource="#
       MessageLevelSecurity"/>
   <security:requiresCredential rdf:resource="#
       Timestamp"/>
   <security:hasSignatureAlgorithm rdf:resource="#
       SHA-1"/>
   <security:signsPart rdf:resource="#Body"/>
  </security:XML-Dsig>
  <security:XML-Dsig rdf:ID="requirement1">
   <security:hasSecurityScope rdf:resource="#
       MessageLevelSecurity"/>
   <security:requiresCredential rdf:resource="#
       Timestamp"/>
   <security:hasSignatureAlgorithm rdf:resource="#
       SHA-1"/>
   <security:signsPart rdf:resource="#Body"/>
  </security:XML-Dsig>
 </wsp:All>
 </wsp:ExactlyOne>
</wsp:Policy>
```

The requirements are the XML-Encryption and the XML-DigitalSignature protocols, which are also described with some properties imposing the specific encryption algorithm to be used and the need to encrypt and sign the Body of the message.

The capabilities are the XML-Enc, XML-Dsig and the Authentication assertions. The first two are similar to what is already expressed into the requirements, except for some specific algorithms (the latter represents a security objective ensured by the X.509 certificate).

Requestor policy. The service requestor expresses its policy in terms of requirements and security capabilities too. In order to show the effectiveness of the semantic matching algorithm, it was manually created from the service provider policy, by inverting requirements with capabilities. Then, some elements and properties have been modified in order to show the reasoning capabilities of the framework. In particular:

- the first requirement was changed into the SAML protocol, that is, into a more specific requirement;
- in the second requirement, the hasSecurityScope property was changed into CommunicationSecurity;
- the third requirement and first capability were left unchanged;
- in the second capability, the hasSignatureAlgorithm property was changed into hasAlgorithm, that is, into a less specific property.

5.1.2. Matching results

The matching algorithm tries to combine the client requirements with the server capabilities (and vice versa) for each couple of policy alternatives. Since in this example there is only one alternative, there are no iterations among alternatives. The match levels found by the reasoner are the following:

Service requirements versus requestor capabilities

- *Possible match*: it derives from the comparison between the Authentication and SAML assertions, since the requirement is more specific than the capability.
- *Close match*: it derives from the comparison between the XML-Enc assertions which are identical in both policies, but have different properties. In particular the requirement property CommunicationSecurity is more general than the MessageLevelSecurity property of the capability.
- *Perfect match*: it derives from the comparison between the XML-Dsig assertions which are identical in both policies, and also have identical properties.

Service capabilities versus requestor requirements

- *Perfect match*: it derives from the comparison between the XML-Enc assertions which are identical in both policies, and also have identical properties.

- *Possible match*: it derives from the comparison between the XML-Dsig assertions which are identical in both policies, but have different properties. In particular, the requirement property hasSignatureAlgorithm is more specific than the capability property hasAlgorithm.

The policies' overall match level is given by the lowest among the found levels: in this case it is a *Possible match*. The Matchmaker communicates to the client that the selected service may satisfy its security requirements. Since the match is not Perfect, the client is required to provide additional information or negotiate with the service provider before actually invoking the service. Further details on the negotiation phase that may carried out at this step can be found in [23].

It should be noticed that, if a traditional (syntactic) approach had been used for the policy comparison, the result would have led us to consider the two policies as incompatible.

5.2. Case study 2

The second case study describes a simple cloud scenario, where a customer is looking for a pool of VMs (virtual cluster) together with some amount of cloud storage. Regarding security concerns, the customer wants to select a provider offering an adequate level of confidentiality for the data storage as well as for the internal communication of VMs belonging to the virtual cluster. In addition, a strong level of authentication is required for remote access of the virtual cluster. Two cloud providers (*CP1* and *CP2*) are available with slightly different security policies: the example will show how the framework allows to select the one with a better matching with the customer security requirements. In this example, let us assume that security policies are already annotated by using semantic concepts belonging to the proposed security ontology.

5.3. Security policies definition

The definition of security policies for both the customer requirements and the cloud providers 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.

Customer policy. The Customer expresses its policy in terms of requirements and security capabilities. The former is the security requirements imposed on cloud providers, while the latter expresses the security capabilities that the Customer is able to offer. Let us note that, if the *Target* concept is not specified in the policy, the requirement (or the capability) refers to the whole interaction between the customer and the cloud provider. In our example the client defines the following:

- *Customer security requirements*
  - Authentication and confidentiality for accessing the service
  - Strong confidentiality (with AES algorithm) for data storage
  - Strong confidentiality (with AES algorithm) for internal virtual cluster communication.
- *Customer 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="sec-ns#">
<wsp:ExactlyOne>
 <wsp:All>
   <security:Protocol rdf:ID="requirement0">
     <security:hasObjective
       rdf:resource="sec-ns#Authentication"
       rdf:resource="sec-ns#Confidentiality"/>
   </security:Protocol>
   <security:Confidentiality rdf:ID="requirement1">
     <security:hasTarget
       rdf:resource="mosaic-ns#DataStorage"/>
     <security:hasEncryptionAlgorithm
       rdf:resource="sec-ns#AES"/>
   </security:Confidentiality>
   <security:Confidentiality rdf:ID="requirement2">
     <security:hasTarget
       rdf:resource="mosaic-ns#Networking"/>
     <security:hasEncryptionAlgorithm
       rdf:resource="sec-ns#AES"/>
   </security:Confidentiality>
   <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>
```

Cloud providers policies. The cloud providers express their security policies by defining requirements and capabilities too. In our example the first cloud provider (*CP1*) is supposed to define the following:

- *CP1 security requirements*
  - Customer authentication with X509 certificate.
- *CP1 security capabilities*
  - HTTPS for customer access
  - Encryption of data storage (with AES-256 algorithm)
  - Encryption of internal VMs communication (with AES-256 algorithm).

The policy of CP1 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="sec-ns#">
<wsp:ExactlyOne>
 <wsp:All>
   <security:Authentication rdf:ID="requirement0">
     <security:requiresCredential
       rdf:resource="sec-ns#X.509"/>
   </security:Authentication>
   <security:Https rdf:ID="capability0" >
     <security:hasSecurityScope
       rdf:resource="sec-ns#TransportLevelSecurity"/>
     <security:requiresCredential
       rdf:resource="sec-ns#X.509"/>
     <security:hasEncryptionAlgorithm
       rdf:resource="sec-ns#AES-256"/>
     <security:hasSignatureAlgorithm
       rdf:resource="sec-ns#SHA-1"/>
   </security:Https>
   <security:Confidentiality rdf:ID="capability1">
     <security:hasTarget
       rdf:resource="mosaic-ns#DataStorage"/>
     <security:hasEncryptionAlgorithm
       rdf:resource="sec-ns#AES-256"/>
   </security:Confidentiality>
   <security:Confidentiality rdf:ID="capability2">
     <security:hasTarget
       rdf:resource="mosaic-ns#Networking"/>
     <security:hasEncryptionAlgorithm
       rdf:resource="sec-ns#AES-256"/>
   </security:Confidentiality>
 </wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
```

Table 2
Customer requirements and Cloud providers capabilities matching.

Customer requirements	CP1 capabilities	CP1 match level	CP2 capabilities	CP2 match level
Authentication and confidentiality	HTTPS	Close match	HTTPS	Close match
Strong confidentiality (AES) for data storage	AES-256 encryption algorithm	Close match	AES-128 encryption algorithm	Close match
Strong confidentiality (AES) for VMs communication	AES-256 encryption algorithm	Close match	3DES encryption algorithm	No match

Table 3
Cloud providers requirements and Customer capabilities matching.

Customer capabilities	CP1 requirements	CP1 match level	CP2 requirements	CP2 match level
SAML with X.509 certificate	Customer authentication with X.509 certificate	Close match	Customer authentication with X.509 certificate	Close match

The second cloud provider (*CP2*) defines a security policy with the same requirements of the first one, but with the third different capability:

- *CP2 security capabilities*
  - HTTPS for customer access
  - encryption of data storage (with AES-128 algorithm)
  - encryption of internal VMs communication (with 3DES algorithm).

5.4. Matching algorithm results

The matching algorithm tries to combine the Customer requirements with the cloud providers capabilities (and vice versa) by following the approach described in Section 4.3:

- The customer's requirement of authentication and confidentiality for accessing the service is compared to the HTTPS capability supported by both cloud providers: in both cases the resulting level of match is *Close Match* because from the ontology the *Reasoner* infers that the HTTPS protocol supports both the Authentication objective than the Confidentiality one.
- Both cloud providers express a requirement related to a strong Customer authentication: it is satisfied by the Customer capability to use SAML with X.509 certificate for authentication purposes. The resulting match level is again *Close Match* in both cases, because the *Reasoner* infers that the objective of authentication is satisfied by the SAML protocol.
- The Customer requirement about strong confidentiality for the Target *DataStorage* is compared with the corresponding capabilities offered by the cloud providers on the same target. The resulting match level with both providers is *Close Match*, since the confidentiality requirement is satisfied with the same encryption algorithm.
- The Customer requirement about strong confidentiality for the Target *Networking* is compared with the corresponding capabilities offered by the providers on the same target. In this case the result is different for the two providers: the resulting match level with CP1 is *Close Match*, while the result with CP2 is *No Match* because the algorithm required by the Customer is stronger than the one offered by CP2.

The results are summarized in Tables 2 and 3, that respectively show the match of Customer requirements versus cloud provider capabilities and the match of cloud provider requirements versus Customer capabilities. According to what is already described in Section 4.3, the resulting match level is the minimum among the individual match levels evaluated for each requirement-capability pair: in this case, this means that the overall match level with CP1 is *Close Match*, while the one with CP2 is *No Match*.

The Matchmaker finally communicates these results to the Customer, informing it that the cloud provider CP1 can satisfy its security requirements, while CP2 cannot.

6. Conclusion

Cloud computing has emerged as an effective technology to build complex service composition scenarios. But, in order to achieve a proper end-to-end security level in these heterogeneous environments, expressing, modeling and matching security requirements and capabilities need to leverage on formal and interoperable mechanisms and languages.

In this paper, after showing the limitation of syntactic-only approach for expressing security policies, we proposed to leverage on semantic technologies, enabling a flexible and powerful matchmaking process between customers and providers security requirements. A security feature can be modeled within a policy, by using terms and concepts defined in a proposed security ontology. The semantic framework is also compatible with syntactic policies defined through the WS-SecurityPolicy specification. A procedure was devised to transform the policy's syntactic terms into semantic concepts of our security ontology. Finally, in the paper we describe a prototype implementation of the framework, whose effectiveness has been validated by means of two different use-case test scenarios.

In the future work, since the developed ontology supports just the main security concepts, we plan to integrate the rest of concepts. Also, intensive tests will be conducted on the system prototype to prove that the implemented semantic engine is able to scale.

References

[1] H. Takabi, J.B. Joshi, G.-J. Ahn, Security and privacy challenges in cloud computing environments, IEEE Secur. Privacy 8 (6) (2010) 24–31.
[2] T. Phan, J. Han, J. Schneider, T. Ebringer, T. Rogers, A survey of policy-based management approaches for service oriented systems, in: 19th Australian Conference on Software Engineering, ASWEC 2008, 2008, pp. 392–401.
[3] OASIS, WS-SecurityPolicy 1.3, OASIS Standard, 2012. Available at: http://www.oasis-open.org/specs/. URL: http://www.oasis-open.org/specs/.
[4] W3C, Web Services Policy 1.5—Framework, W3C Recommendation, September 2007. Available at: http://www.w3.org/TR/ws-policy/. URL: http://www.w3.org/TR/ws-policy/.
[5] G. Di Modica, O. Tomarchio, A semantic model to support security matching in cloud environments, in: CLOSER 2013—Proceedings of the 3rd International Conference on Cloud Computing and Services Science, Aachen, Germany, 2013, pp. 427–436.
[6] G. Tonti, J. Bradshaw, R. Jeffers, R. Montanari, N. Suri, A. Uszok, Semantic Web languages for policy representation and reasoning: a comparison of KAoS, Rei, and Ponder, in: International Semantic Web Conference, (ISWC2003), Springer, Florida, USA, 2003, pp. 419–437.
[7] N. Damianou, N. Dulay, E. Lupu, M. Sloman, The Ponder policy specification language, in: Proceedings of the International Workshop on Policies for Distributed Systems and Networks, POLICY'01, Springer-Verlag, London, UK, 2001, pp. 18–38.
[8] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, 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, POLICY'03, IEEE Computer Society, Washington, DC, USA, 2003, pp. 93–96.

[9] L. Kagal, T. Finin, 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, POLICY'03, IEEE Computer Society, Washington, DC, USA, 2003, pp. 63–74.
[10] N. Sriharee, T. Senivongse, K. Verma, 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, 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, L. Hai-guang, Semantic security policy for web service, in: 2009 IEEE International Symposium on Parallel and Distributed Processing with Applications, 2009, pp. 258–262.
[14] W3C, Web Services Description Language (WSDL) 2.0, W3C Recommendation, June 2007. Available at: http://www.w3.org/TR/wsdl20/.
URL: http://www.w3.org/TR/wsdl20/.
[15] D.Z.G.A. Garcia, 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, 2008, pp. 331–338.
[16] A. Kim, J. Luo, 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] G. Di Modica, O. Tomarchio, Semantic annotations for security policy matching in WS-Policy, in: SECRYPT 2011—Proceedings of the International Conference on Security and Cryptography, Seville, Spain, 2011, pp. 443–449.
[18] G. Di Modica, O. Tomarchio, Semantic security policy matching in service oriented architectures, in: Proceedings—2011 IEEE World Congress on Services, SERVICES 2011, Washington, DC, USA, 2011, pp. 399–405.
[19] F. Moscato, R. Aversa, B. Di Martino, T. Fortis, V. Munteanu, An analysis of mOSAIC ontology for Cloud resources annotation, in: 2011 Federated Conference on Computer Science and Information Systems, FedCSIS, 2011, pp. 973–980.
[20] The Open Grid Forum, The Open Cloud Computing Interface, 2011. http://occi-wg.org/.
[21] M. Paolucci, T. Kawamura, T.R. Payne, K.P. Sycara, Semantic matching of web services capabilities, in: ISWC'02: Proceedings of the First International Semantic Web Conference on the Semantic Web, Springer-Verlag, London, UK, 2002, pp. 333–347.
[22] G. Di Modica, O. Tomarchio, A semantic discovery frame work to support supply–demand matchmaking in cloud service markets, in: CLOSER 2012—Proceedings of the 2nd International Conference on Cloud Computing and Services Science, Porto, Portugal, 2012, pp. 533–541.
[23] L. Liccardo, M. Rak, G. Di Modica, O. Tomarchio, Ontology-based Negotiation of security requirements in cloud, in: Proceedings of the 2012 4th International Conference on Computational Aspects of Social Networks, CASoN 2012, Sao Carlos, Brasil, 2012, pp. 192–197.

Giuseppe Di Modica received his Degree in Computer Engineering in 2000 from the Engineering Faculty of University of Catania (Italy). Since then he has been engaged in research on distributed systems with the Department of Computer and Telecommunications Engineering of Catania University. In 2005 he received the Ph.D. in Computer Science and Telecommunication Engineering from University of Catania. The Ph.D. thesis discussed was entitled "A user-centric analysis of the interworking between heterogeneous wireless systems".

His research interests include mobile agents, ad-hoc networks, wireless sensor networks, mobile computing, Service oriented architectures, Grids and Cloud Computing.

He is currently a research assistant at Engineering Faculty of University of Catania (Italy).

Orazio Tomarchio received his Degree in Computer Engineering from the Engineering Faculty of University of Catania (Italy) in 1995.

In 1999 he received the Ph.D. in Computer Science from University of Palermo. He is currently associate professor with the Department of Electrical, Electronic and Computer Engineering of University of Catania, beginning from February 2002. He has been engaged in teaching several degree and master courses mainly focused on software engineering, security and programming in distributed systems. The scientific activity of Prof. Tomarchio was initially focused on studying distributed systems, particularly with regard to innovative programming and management techniques based on the mobile code paradigm. Recently his scientific interests have been also focused on middleware for mobile ad-hoc network and for personal and ubiquitous computing, mobile P2P computing, Grid and service oriented architectures, security, SLA and market-oriented resource allocation in Cloud computing, and semantic technologies.