Security Annotation for DAML Web Services*

Grit Denker[1] and Lalana Kagal[2]
[1]SRI International
Menlo Park, California
denker@csl.sri.com
[2]University of Maryland Baltimore County
Baltimore, Maryland
lkagal1@cs.umbc.edu

Abstract

The focus of this paper is developing ontologies that can be used to annotate web services represented by DAML-S. We propose several security-related ontologies that are designed to represent security standards such as XML Signatures in terms of their characteristics like credentials, mechanisms supported, notations used, etc. These ontologies are used to describe security properties of web services, agents and users. These properties can be specific by stating the particular standards/protocols supported or more general in terms of the security mechanisms used, the credentials required or notations specified. The security properties associated with registered web services as well as requests (originating from other services, agents and human operators) for web services are security requirements and capabilities. A reasoning engine decides whether a web service satisfies a request by comparing security characteristics. The requirements of the request need to be satisfied by the capabilities of the potentially matching web service, whose requirements need to be satisfied by the capabilities specified in the request (which could represent the capabilities of the agent which makes the request). Our prototypical implementation uses JTP, the Java Theorem Prover from Stanford, for deciding the degree to which the requirements and capabilities match based on our matching algorithm.

1 Introduction

Though today's Internet is a vast information resource, its lack of structure and metadata make it difficult to extract the desired information in a reasonable time. The Semantic Web is a vision of a future Internet in which web resources are enriched with machine-processable metadata, which describes their meaning. This metadata will enable software agents or search engines to find and interpret web content much more quickly and precisely than is possible with current techniques, such as keyword search or data mining. In the Semantic Web, resources can be evaluated with respect to their appropriateness for the given query, which in turn will lead to greater efficiency of web resource allocation, despite the daily expansion of web space. The DARPA Agent Markup Language (DAML [4]) is a language that allows the annotation of web pages to indicate their meaning. Its latest version, DAML+OIL [3], is a combination of DAML with the European OIL (Ontology Inference Layer) [13], which in turn is built on top of the description logics [10]. One of the advantages of DAML+OIL over other markup languages like XML or RDF is its expressiveness through built-in semantic concepts. For instance, DAML+OIL allows the definition of relationships between classes such as inheritance (subclassing), equivalence, or construction of classes as boolean combinations of other classes (e.g., intersection or union of classes). These syntactical features allow the capture of relevant semantic information in an ontology that proves useful when reasoning about ontologies and web resources that are marked-up with such ontologies.

In this paper, we are bridging the gap between the Semantic Web and security through DAMLized security annotations and by providing brokering over these annotations. Information security plays an increasingly critical

*Supported by the Defense Advanced Research Projects Agency through the Air Force Research Laboratory under Contract F30602-00-C-0168.

role in society. Over the last decade there has been considerable progress in the field of security, ranging from theoretical foundations and models, over languages, methods, and tools, to applications and security system development. Given the increased importance of the World Wide Web for business, industry, finance, education, government and other sectors, security will play a vital role in the success of the Semantic Web. It is essential to have tools and techniques in place that will allow the storage, maintenance, and processing information of the Semantic Web in ways that meet security requirements such as authentication, authorization, and data integrity among others.

Our work focuses on security aspects for DAML web services. DAML-S [7, 5] is a language that supports a set of basic classes and properties (defined in DAML+OIL) for declaring and describing services. This work has its roots in previous work about web services [15, 6, 11, 12, 9]. Its top-level class is `Service`. A service presents a *ServiceProfile* (what it does), is described by a *ServiceModel* (how it works), and supports *ServiceGroundings* (how it is used). DAML-S is used for describing capabilities of services and procedures available on the Web.

In this paper, we aim to provide a framework that will allow the annotation of web services and agents with security information on a very high abstraction level. Our work aims to provide a layer of abstraction on top of the various existing security-related standards (such as XML Signature [16, 2]) and other work that is currently under development (such as WS-Security [1]). We propose several security-related ontologies that are designed to represent well-known security techniques in terms of their characteristics like credentials, mechanisms supported, notations used, etc. In particular, we propose ontological elements for expressing security aspects of web services and show how such annotations can be exploited to make matchmaking decisions in situations where agents search for web services with certain security characteristics. Our security ontologies are defined in DAML+OIL making the integration with DAML-S straight-forward. These ontologies are used to describe the security requirements and capabilities of web services and requesting agents. A reasoning engine decides whether agents and web service have comparable security characteristics by verifying that the agent's requirements are satisfied by the web service's capabilities and the service's requirements are met by the agent's capabilities. Our prototypical implementation uses JTP, the Java Theorem Prover from Stanford [8], for deciding the degree to which the requirements and capabilities match based on our matching algorithm.

We begin with a motivating example and the overview of our work in Section 2 that sets the context of our work and also defines its boundaries. Section 3 introduces high-level ontologies for credentials and various security mechanisms, such as encryption, digital signatures, etc. Moreover, we link our ontology to the existing standards such as PGP, XML Signatures, etc. Security markup adds value to the semantic web when used in connection with inference systems that support the process of deciding which web services matches a request. Here the inherent semantic meaning of DAML ontologies comes into play. In Section 4 we describe the algorithm used to decide whether two security descriptions are related and discuss the implementation of the security reasoner.

Note: The ontologies discussed in this paper can be found at `www.csl.sri.com/users/denker/daml-sec`. We provide a credential ontology (`www.csl.sri.com/users/denker/daml-sec/credential.daml`), an ontology with general security notations (`www.csl.sri.com/users/denker/daml-sec/security.daml`) and an ontology which proposes how to make use of the former two ontologies in the context of DAML-S web services (`www.csl.sri.com/users/denker/daml-sec/service-sec.daml`). We define additional parameters for service profiles that allow to express security related information. In this sense, our current approach is loosely connected to DAML-S. That means, in order to be able to reason about security for DAML-S services, one would have to load the security ontologies along with other application-specific ontologies. Whether security parameters will be become first-class citizen of DAML-S profiles is subject to future discussion.

2 Overview and Motivation

Our work is targeted towards situations in which agents[1] and web services have security markup as well as other more functionally-oriented markup. An agent has the task to find a web service with a specific functionality.

[1] We use the term agent to represent any requesting entity, including an agent, a human user or another service

Additionally, the agent is interested only in those web services that fulfill certain security properties, such as communicating via encrypted messages without needing authentication, to name an example. The agent itself has capabilities such as the credentials it holds or protocols it is able to use, that will determine which web services are a possible match for the agent's request. Similarly, a web service has capabilities including which security mechanism it utilizes, which credentials it is able to accept etc. Along with capabilities, a web service may also have its own requirements on agents that it is willing to communicate with. For example, a web service might have the capability to sign all outgoing messages and it might require subscribing agents to authenticate using a login. Therefore, though the web service may provide the functional capabilities that the agent is looking for (for instance, an online reservation service), the web service and agent may not match in terms of their security requirements and capabilities.

Our work defines necessary notations to express security-related capabilities and requirements of web services, so that they can be exploited by a special-purpose reasoner and matched against agent requests. Web services register with the Matchmaker service, describing both their functional capabilities (such as name, parameters, etc.) as well as security-related information. The ontologies we suggest in Section 3 can be used to mark-up web services with respect to security in terms of mechanisms, credentials etc. Examples for security requirements are ”authentication by X.509 certificates” or ”use of SSH protocol”. Examples for security capabilities are ”possession of a login” or ”possibility to authenticate oneself.” An agent making a request essentially fills out a web service template, describing its “dream”-service. Along with describing its own security capabilities, it includes the capabilities it desires in the service as its security requirements. The request is sent to the Matchmaker, who after finding a list of services that meet the functional requirements of the agent, will utilize the security reasoner to decide the sub-problem; whether the agent's request match the capabilities and requirements of any web service in the list.

Here are some examples how one can make use of security capability and requirement markups. For our work we are assuming that the Matchmaker is trusted.

Example 1 : An agent A is looking for a travel web service. Agent A, using the Matchmaker interface, fills out a template describing the desired functionality of the web service as well as the agent's security requirements and capabilities. Let's assume that the agent is only capable of performing Open-PGP encryption and requires that the travel service be of capable of authenticating itself with a X.509 certificate.

A travel web service T registers with the same Matchmaker. It provides its name, description, normal functional capabilities, security requirements and security capabilities. We assume that the travel service requires an agent be able to perform encryption mechanism and the service itself is capable of the XKMS protocol for message exchanges.

When the agent submits its request, the Matchmaker goes through the description of all services registered with it to find a set of services that provide travel functionality. So, the matchmaker finds service T as a functional match and checks the security requirements and capabilities of agent A against those of the web service T. In this case there is a match, because the agent's requirements are fulfilled by the service's capabilities and the service's requirements are met by the agent's capabilities. But what happens if the security capabilities and the security requirements are not subsumptions of each other? This brings us to the next scenario.

Example 2 : In this scenario, the agent is capable of encryption but the web service requires Open-PGP encryption. The requirement of the web service is a stronger condition than what can be asserted by the capabilities of the agent and thus, there is no match. One could imagine that a negotiation phase will be entered. It is possible that the agent did not register its full capabilities with the matchmaker and might be willing to disclose more of its capabilities once it starts negotiating. Negotiation could be handled by a parallel thread of the Matchmaker. It contacts one or both of the parties that had an almost matched pair and asks the agent/service with the more non-specific capability/requirements if they are willing to do the more specific protocol/mechanism etc. If the entity replies positively, the matchmaker informs the agent that a match was found. This negotiation may have to be part of the Matchmaker because it may not be possible for the “almost matched” agent and service to communicate with each other as they may not know each others security protocol for communication.

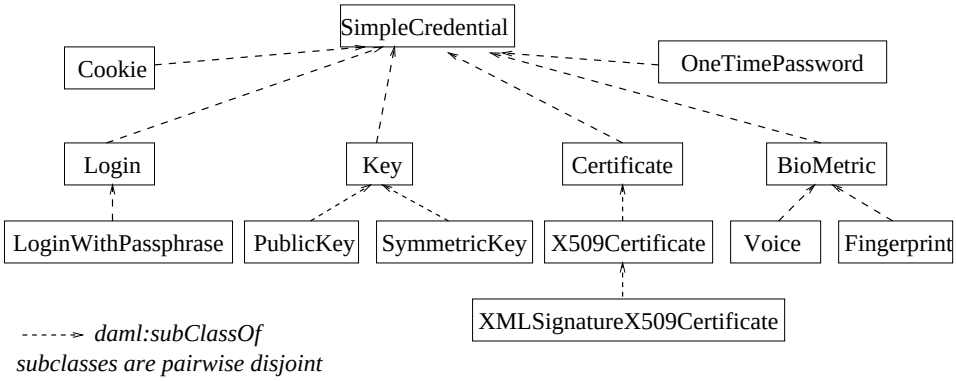

Figure 1: A credential ontology (class hierarchy I)

We provide ontologies that are the basis for doing automatic subsumption reasoning over security annotations. Thus, instead of matchmakers being modified for every new security protocol or standard or mechanism that comes along and having the matchmakers do special case implementations, we provide a framework that is capable of doing automatic reasoning over security characteristics of web services and requests about them.

3 DAML Ontologies

In the Semantic Web of the future, web resources can be annotated with respect to their content and meaning. One particular application of semantic markups is the specification of security information concerning access restrictions and requirements as well as reliability assertions of web services. Our ontology comprises notations to markup web services with respect to access control, data integrity, authorization, protocols and more. A web service may have restricted access. In order to be granted access to the service, the requesting agent or user has to pass an authorization test. Traditionally, authentication techniques are used to get authorized access. A web services may provide assertions about the integrity of message exchange, for instance, by using encrypted communication channels to avoid that data has been inadvertently or deliberately altered or by using specific protocols. The use of cryptographic techniques assures the integrity of stored or received data. Similarly, an agent has provisions and requirements with respect to security. An agent might be in possession of a certificate that it can use for authentication or it might be given the requirement to find a web service that is able to sign sent messages.

Our goal is to define security ontologies in DAML+OIL [3] that allow to annotate agents and web services with respect to various security related notions such as access control, data integrity and others. We start with an ontology that summarizes various ways in which authentication using credentials can take place.

3.1 Credentials

The process of establishing and verifying the identity of a requesting party in a web application, so-called authentication, is often the basis of the decision whether access is granted or not. Authentication is based on some token, also called credential, that the requesting party would know or have. Credentials can employ different well-know authentication techniques such as name-passphrase login, public and private keys or certificates. Our goal is to be able to specify access control restrictions of web pages or web services that use authentication as a requirement for authorized access. We envision security policies that describe access restrictions on conceptual level. The following are examples of some rules that may be part of an access policy:

- A webpage that is access-protected by means of certificates can only be read by a user when an appropriate certificate is presented.

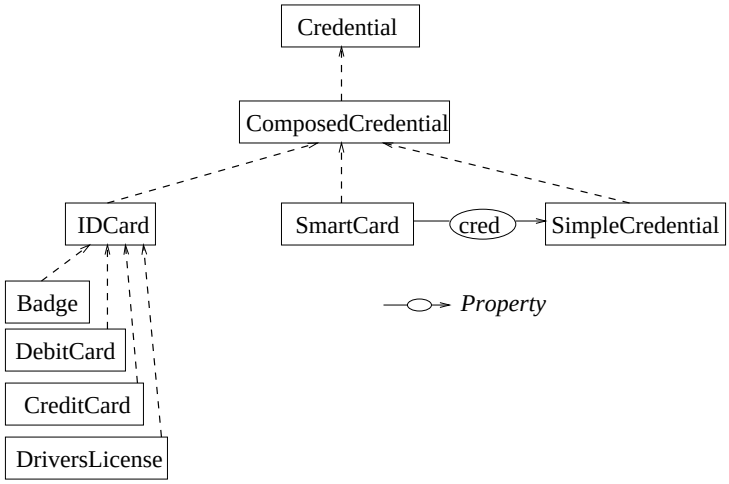

Figure 2: A credential ontology (class hierarchy II)

- Users without an appropriate type of credential are not allowed access to protected resources.
- A resource that is protected by means of a login requires a user name (or a similar concept such as an identity) and a passphrase (or a similar concept such as a password).

We would like to point out one particular aspect of policy rules. Rules usually are meant to comprise in a succinct way various possible instantiations. For instance, the last of the three rules expresses that a resource that is restricted by a login requires two input fields from the user, one being the user name or identity, and the other the password. The rule also intends to capture that the precise "labels" of the input fields are not crucial, as long as the two fields have the intended meaning (e.g., password vs passphrase). The second rule states that a resource that is protected by means of a specific kind of credential (such as X509 certificates) can not be accessed with any other kind of credential (such as login). Thus, this rule says that all other types of credentials are not valid (without naming those types of credentials explicitly).

We aim at providing means to mark up web resources with access policies similar to the above ones. Different types of credentials are at the core of access restrictions. Thus, our first ontology in DAML+OIL defines the kind of credentials that are most commonly used in today's internet security.

We distinguish between "SimpleCredential" and "ComposedCredential". The top-level class "SimpleCredential" (see Figure 1) is subclassed to "Cookie, Login, Key, Certificate, BioMetric", and 'OneTimePassword" (subclass relationships are depicted using dotted arrows). All subclasses are pairwise disjoint. "Public Key" and "Symmetric Key" are disjoint subclasses of the key class. The certificate class is specialized to "X509Certificate", and further to the specific class of X509 certificates in the XML Signature [2].

We have defined some of the most commonly existing classes of composed credentials (see Figure 2, such as "IDCard" and "SmartCard". For example, a smart card can contain data such as keys, biometric templates or PINs for authentication. Thus, composed credentials often contain simple credentials, as modeled in our ontology with a property "cred". Various specializations of identity cards are given. A simple credential is also a subclass of the composed credential class. Our ontology is extensible to allow for more credential classes or further properties.

The above figures only depict class and their inheritance relationships. Properties and other restrictions are defined in the ontologies as well (see `www.csl.sri.com/denker/daml-sec/` for complete ontologies). For example (see Figure 3) the login class has two datatype properties defined, "name" and "passphrase", both of type string. We are using the DAML "restriction" concept to express that the cardinality on these properties for the login class is constrained to one. That means, each login credential comes along with exactly one name and one passphrase. For the certificate class we defined an object property, that is a property with a DAML class as its range type. The property "assoc" associates with each certificate some data (such as issuer name, serial

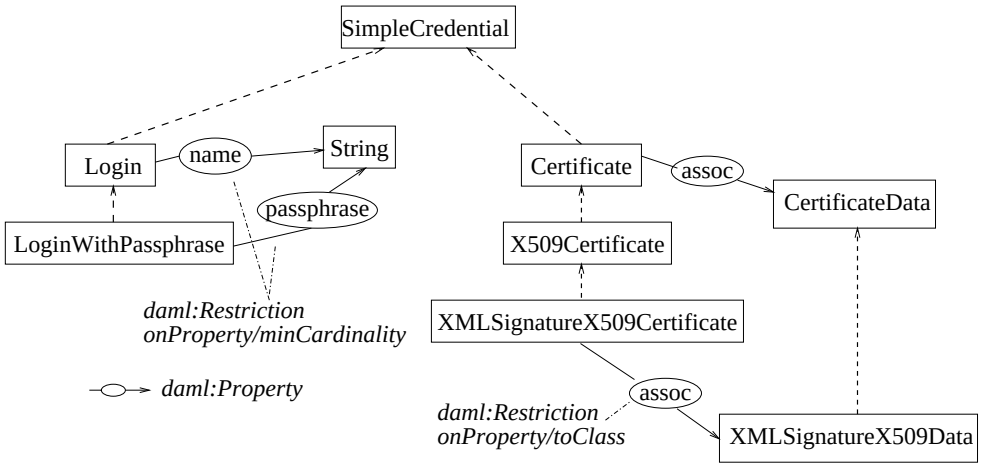

Figure 3: A credential ontology (some properties)

number, etc). In the special case of an XML Signature certificate, the object property is restricted to be of type "X509Data". This class is defined to be equivalent to the X509 element of the XML Signature definition in [2]. This way we tie our ontology into an ontology that is being standardized.

The tie into the XMLSignature specification [2] is useful to express more detailed security policies. For instance, one can imagine situations in which a web page only accepts X509 certificates that have been issued by a particular certification authority such as VeriSign or Thawte. Such information will be extremely helpful in directing software agents to web resources that are available to them. A software agent that searches for a particular web service could be equipped with a collection of certificates. Whenever the agent encounters a service that satisfies other functional aspects of its request, it may do some simple computation to conclude whether the service will be available.

XML Signature is an emerging standard that allows to express certificate details such as issuer. This standard that was jointly proposed by IETF and W3C defines how signatures on any digital content can be created and represented in XML. Digital signatures are often used to ensure that data has not been subject to alterations, inadvertently caused by faulty transmission channels or deliberately caused by intrusive behaviors. In particular, the XML Signature framework includes references to well-known cryptographic algorithms and key management systems. A `KeyInfo` element in an XML `Signature` element indicates the keys that need to be used in order to validate the signature. XML Signature has syntax elements to define key data related to X509 certificates [2], to PGP public key pairs [17], and SPKI public key pairs [14]. We can exploit those structures by defining specific instance of the class `Credential` in our security ontology. For instance, a certificate that complies with the syntax of the XML Signature X509 key data element can be defined as follows:

```
<daml:ObjectProperty rdf:ID="assoc">
  <daml:range rdf:resource="CertificateData"/>
</daml:ObjectProperty>

<daml:Class rdf:ID="XMLSignatureX509v3Certificate">
  <daml:subClassOf rdf:resource="#Certificate">
  <daml:Restriction daml:cardinality="1">
    <daml:onProperty rdf:resource="#assoc"/>
    <daml:hasClass rdf:resource="http://www.w3.org/2000/09/xmldsig#X509Data"/>
  </daml:Restriction>
</daml:Class>
```

3.2 Security Mechanisms

Over the last couple of years many security-related frameworks for web applications have been proposed. A fair number of them are based on XML, like XML Signature, SAML, XACML, and WS-Security. Besides those approaches there are numerous well-known and widely adopted mechanism for security, ranging from abstract notions such as encryption and digital signatures implemented in various standards such as Open-PGP to more specific protocols such as SSH and IKE.

To our best knowledge, no integrating framework for this wide array of security-related approaches exists. The openness of the semantic web dictates that there will be no such thing like the one standard for security that will be adopted. Rather we expect that new protocols or mechanisms for security will emerge as research progresses. Nevertheless, we think that languages like DAML+OIL could be used to provide bridges between different formalism and enable interoperability. This is our motivation in providing a security ontology that is able to describe security mechanism of various different kinds on a very high abstraction level. Another advantage of using an ontological approach and a language like DAML+OIL is, that our approach is extensible. As new mechanism become available, we can extend the existing classes and instances in order to incorporate the latest developments. Our current ontology is not meant to be complete, we rather aimed to capture the most widely used security mechanism and ensure the possibility of need-driven extensibility.

We propose an ontology that allows to interface on a high level of abstraction among various security standards and notations.

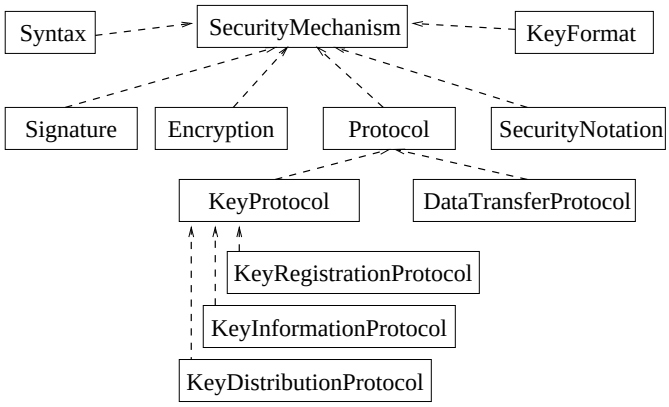

Figure 4: A security ontology I

Several properties are defined for the top class "'SecurityMechanism" (not shown in Figure). For example, the ontology defines an object property "syntax" that has the class "Syntax" as range, another property "relSecNotation" has the class "SecurityNotation" as its range, and "reqCredential" has the credential class as range. There are various instances for the defined classes. To name a few, instances of syntax are "ASCII, DAML+OIL, OWL, XML, MIME", security notations are "Authentication, Authorization, AccessControl, DataIntegrity, Confidentiality, Privacy, ExposureControl, Anonymity, Negotiation, Policy, KeyDistribution," and "X.509" is an instance of the KeyFormat class. Protocols such as "X-KRSS, X-KISS, Kerberos," or "SOAP" are defined as instances of the appropriate protocol subclasses. "XML-DSIG" is an instance of the signature class and "OpenPGP-Enc" is of type "Encryption."

In combination, the subclasses of "SecurityMechanism" with their instances and the defined properties can be used to describe security characteristics. For example, a web service request may state that the agent requires the use of X.509 certificates or, more generally, an agent may request a service that provides some form of authentication.

To express these (and other) security requirements, we introduce special restriction classes that will later be used to characterize an agent's request. Restriction classes are classes in which we constrain the range of one of the object properties "reqCredential, syntax, relSecNotation" etc. Below are examples of such restriction classes.

- Restriction classes that have specific relSecurityNotations

```
<daml:restriction rdf:ID="AuthenticationSubClass">
     <daml:onProperty rdf:resource="#relSecNotation"/>
     <daml:hasValue rdf:resource="#Authentication"/>
</daml:restriction>

<daml:restriction rdf:ID="AuthorizationSubClass">
     <daml:onProperty rdf:resource="#relSecNotation"/>
     <daml:hasValue rdf:resource="#Authorization"/>
</daml:restriction>
```

- Restriction classes that have specific reqCredentials

```
<daml:restriction rdf:ID="LoginSubClass">
  <daml:onProperty rdf:resource="#reqCredential"/>
  <daml:hasValue rdf:resource="&credential;#Login"/>
</daml:restriction>

<daml:restriction rdf:ID="X509CertificateSubClass">
  <daml:onProperty rdf:resource="#reqCredential"/>
  <daml:hasValue rdf:resource="&credential;#X509Certificate"/>
</daml:restriction>
```

  If a web service has a related security notation of type “Authentication”, then it is also of type “AuthenticationSubClass”. A specific web service can satisfy many restriction classes. The matching criteria is to match at least all restrictions requested by the agent.

- Restriction classes that have specific syntax

```
<daml:restriction rdf:ID="ASCIISubClass">
  <daml:onProperty rdf:resource="#syntax"/>
  <daml:hasValue rdf:resource="#ASCII"/>
</daml:restriction>

<daml:restriction rdf:ID="DAML+OILSubClass">
  <daml:onProperty rdf:resource="#syntax"/>
  <daml:hasValue rdf:resource="#DAML+OIL"/>
</daml:restriction>

<daml:restriction rdf:ID="OWLSubClass">
  <daml:onProperty rdf:resource="#syntax"/>
  <daml:hasValue rdf:resource="#OWL"/>
</daml:restriction>
```

Specific protocols can be defined using some of the characteristics they have. For example, XKMS is specified as follows in our ontology.

```
<security:KeyProtocol rdf:ID="XKMS">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:ID="#AuthenticationSubClass"/>
    <daml:Class rdf:ID="#KeyDistributionSubClass"/>
    <daml:Class rdf:ID="#XMLSubClass"/>
  </daml:intersectionOf>
  <security:documentation rdf:resource="#XKMS-Ref"/>
</security:KeyProtocol>
```

The missing link between our security ontologies and web services is introduced below. We define a SecurityMechanism to be a subclass of ServiceParameter. Then we can declare two new properties for web services, namely “requirement” and “capability” of type SecurityMechanism.

```
<daml:Class rdf:about="&security;#SecurityMechanism">
  <rdfs:subClassOf rdf:resource="&profile;#ServiceParameter"/>
</daml:Class>

<daml:ObjectProperty rdf:ID="securityCapability">
  <daml:subPropertyOf rdf:resource="&profile;#serviceParameter"/>
  <daml:range rdf:resource="&security;#SecurityMechanism"/>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="securityRequirement">
  <daml:subPropertyOf rdf:resource="&profile;#serviceParameter"/>
  <daml:range rdf:resource="&security;#SecurityMechanism"/>
</daml:ObjectProperty>
```

Finally, an agent requesting a web service that is able to do authentication would result in the following “requested web service markup.”

```
<security:KeyProtocol rdf:ID="Sec2">
        <security:relSecNotation rdf:resource="&security;#Authentication" />
</security:KeyProtocol>

<!-- Requirement = Authentication -->
<profile:Profile rdf:ID="RequestServiceProfile1">
  <profile:serviceName>
    <xsd:string>
      <rdf:value>Req: Authentication </rdf:value>
    </xsd:string>
  </profile:serviceName>
  <profile:textDescription>
    <xsd:string>
      <rdf:value>
           This service requires authentication to communicate.
      </rdf:value>
    </xsd:string>
  </profile:textDescription>
  <securityRequirement rdf:resource="#Sec2"/>
</profile:Profile>
```

A registered web service claims to be able to communicate using the XKMS protocol.

```
<security:XKMS rdf:ID="Sec1"/>

<!-- Capability = XKMS -->
<profile:Profile rdf:ID="RegisteredServiceProfile1">
  <profile:serviceName>
    <xsd:string>
      <rdf:value>Capability XKMS</rdf:value>
    </xsd:string>
  </profile:serviceName>
  <profile:textDescription>
    <xsd:string>
      <rdf:value>This service is capable of XKMS communication </rdf:value>
```

```
    </xsd:string>
  </profile:textDescription>
  <securityCapability rdf:resource="#Sec1"/>
</profile:Profile>
```

Given the definition of XKMS, a subsumption reasoner like JTP can resolve that the requirements of the request are fulfilled by the capabilities of the registered web service. The security reasoning algorithm is discussed below.

4 Security Reasoner and Matching Algorithm

Consider Example 1 from Section 2. Agent A, which is capable of performing Open-PGP encryption and requires a communicating service be capable of authenticating itself with a X.509 certificate, makes a request to the Matchmaker for a travel web service. A travel web service T, registered with the same Matchmaker, meets the requirements of the agent A. It requires an agent be capable of an encryption and itself is capable of the XKMS protocol for message exchanges. Our security reasoner accepts as input the requirements and capabilities of the agent and of the service and decides to what degree they match. In this case, the reasoner finds that there is a close match as the capabilities of the service meet the requirements of the agent and the capabilities of the agent meet the requirements of the service.

Requirements and capabilities of agents and services are described using our security ontologies (Please refer to Section 3 for more information). They can be either instances of defined security protocols like XKMS, Open PGP or collections of instantiated characteristics of these protocols like encryption, authentication, or the use of login as a credential. Every agent and service can have more than one requirement and capability. For ease of development, we assume that they are disjunctively related. Adding conjunction does not involve redesign only additional coding. Our security reasoner is in Java and uses Java Theorem Prover (JTP) and our matching algorithm to decide the relationship between the requirements and capabilities. This relationship can be either *perfect match, close match, general match, negotiation possible* or *no match* ranging from best to worst.

The matching algorithm exploits the subsumption capability of JTP to extract the most specific type of a requirement and a capability and then proceeds to match them. The most specific type is the lowest class in the security ontology that the requirement/capability is an instance of. However, the requirement and/or capability need not be of a certain protocol type, and instead be a collection of characteristics associated with protocols. The matching algorithm considers three general cases, when both the requirement and the capability are instances of a protocol/standard, when one of them is an instance of a protocol, and when neither is an instance of a protocol. The algorithm is as follows :

- Case I : Both the requirement and the capability are instances of a security protocol
  - Perfect Match : A capability and a requirement are perfectly matched if they are both instances of the same specific type. For example, if a capability and a requirement are of type XKMS, then there is a perfect match. However, if a capability is of type XKMS and a requirement is of a type which is a subclass of XKMS, then it is not a perfect match.
  - Close Match : If the most specific type of capability is lower in the hierarchy than the most specific type of the requirement, it is said to be a close match. The requirement in this case is more general than the capability. For example, if the requirement is of type XKMS and the capability is of a type which is a subclass of XKMS.
  - Possibility of Negotiation : If the requirement is more specific than the capability, there is a possibility of negotiation as the capability may not adequately represent the entities actual abilities. The most specific type of capability is lower in the hierarchy than the most specific type of the requirement.
  - No Match : If the most specific types of the requirement and capability are not related, then there is no match.

- Case II : Either capability or requirement are instances of a security protocol
  - General match : The capability is an instance of a protocol but the requirement is not. There is a general match if the characteristics of the requirement are a subset of the characteristics of the specific type of the capability. Let us assume that SSH is the most specific type of the capability and the requirement only has notation as authorization, then as the protocol SSH has authorization notation, there is a general match.
  - Possibility of Negotiation : If the requirement has a protocol as its most specific type and the capability does not, then there is a possibility of negotiation if every characteristic of the protocol type of requirement is also a characteristic of the capability. For example, the requirement is Kerberos, which includes authentication and key distribution, and capability has key distribution feature.

- No Match : If there is no general match or a possibility of negotiation in this case, then there is no match.

- Case III : Neither capability or requirement are instances of a security protocol
  - General Match : If the features of the requirement are a subset of the features of the capability. Consider as an example, the requirement includes only authentication and the capability has authentication as one of its features.
  - No Match : If there is no general match in this case, it is considered a no match. If for example, the requirement is for authorization and the capability is authentication and key distribution.

4.1 Extensibility

We include descriptions of several protocols and security characteristics in our ontologies. It is possible to extend these ontologies with other protocols and characteristics, which will be accepted and reasoned over by our framework. As the security reasoner tries to find the most specific type of a requirement or capability or uses the list of features, it provides a general framework that allows our existing ontologies to be extended. New features could either be subtypes of existing features like ecryption2 can be a subclass of encryption or an entirely new unrelated feature, which is simply a subclass of Security Mechanism. Additional protocols will be defined in terms of their features. The reasoner will load these new definitions into JTP and go through the hierarchy of ontologies to find the most specific types of protocols (either predefined or extended) for the requirement/capability or the list of features (either predefined or extended) associated with a requirement/capability and carry out the matching algorithm as described above.

4.2 Walk-Through Example

This section is a walk through two examples of how security annotations of web services and agents are used by our system to provide security specific brokering. The examples demonstrates several features of our security ontology and illustrates the operation of the security reasoner.

Example 1 : We revisit Example 1 from Section 2. Agent A is looking for a travel web service and registers its functional requirements (what it wants the functional description of the matched service to be) and its security requirements (what it expects the security capabilities of a matched service to be) and capabilities (what security functionality it is capable of) with the DAML-S Matchmaker. Agent A is capable of performing OpenPGP encryption and requires a communicating service be capable of authenticating itself and communicating in XML. The following is the a part of the request made by agent A.

```
<security:OpenPGP-Enc rdf:ID="Capability1" />
<security:KeyProtocol rdf:ID="Requirement1">
    <security:relSecNotation rdf:resource="#Authentication" />
    <security:syntax rdf:resource="#XML" />
</security:KeyProtocol>
<Agent rdf:about="#A">
    <securityCapability rdf:resource="#Capability1" />
    <securityRequirement rdf:resource="#Requirement1" />
</Agent>
```

A web service T registers its functional description as travel and its security capabilities (what the service is capable of) as XKMS and its requirements (what it expects communicating agents to use) as encryption. The following is a portion of the description of the service

```
<security:XKMS rdf:ID="Capability2" />
<security:KeyProtocol rdf:ID="Requirement2">
    <security:relSecNotation rdf:resource="Encryption" />
</security:KeyProtocol>
<profile:Profile rdf:about="#T">
    <profile:serviceName> ... </profile:serviceName>
    <profile:textDescription> ... </profile:textDescription>
     <securityCapability rdf:resource="#Capability2" />
```

```
     <securityRequirement rdf:resource="#Requirement2"/>
</profile:Profile>
```

The DAML-S Matchmaker uses the functional requirements of the agent to extract a list of registered agents that match in functionality. Then the Matchmaker uses the security reasoner to decide whether the agent matches any of the services in terms of security characteristics. Both the agent's request and the description of the service are input to the security reasoner. The following steps are taken by the security reasoner to decide whether the agent and service match in terms of their security annotations and to what degree.

- Based on the input the security reasoner makes the following inferences.
  - A's capability has OpenPGP-Enc as the most specific type
  - A's requirement does not have a most specific type and instead has a list of features of a security protocol (authentication and xml)
  - T's capability has XKMS as the most specific type
  - T's requirement does not have a most specific type and is a list of features of a security protocol (encryption)
- The reasoner tries to find the degree of matching between A's requirement and T's capability.
  - As T's capability has a most specific type (XKMS) and A's requirement (authentication and xml) does not, the reasoner selects Case II of the matching algorithm.
  - The reasoner tries General Matching, which is the first case of Case II of the matching algorithm.
  - It locates all the features associated with XKMS, which are authentication, xml and key distribution.
  - The features of the requirement are a subset of the features of XKMS, so General Matching holds.
- The reasoner tries to find the degree of matching between T's requirement and A's capability.
  - As A's capability has a most specific type (OpenPGP-Enc) and T's requirement. (encryption) does not, the reasoner selects Case II of the matching algorithm.
  - The reasoner tries the General Matching case of Case II of the matching algorithm.
  - It locates all the features associated with OpenPGP-Enc, among which is encryption.
  - The features of the requirement are a subset of the features of A's capability, so General Matching holds.
- The security reasoner decides that the agent and service match in terms of their security annotations and the degree is *general match*.

The DAML-S Matchmaker uses the above result of the security reasoner to decide that the agent and the service match in both functionality and security and informs the agent A that the service T *generally matches* its request.

Example 2 : As another example, consider a web service W1 looking for a banking service. It registers its functional description, security capability (SSH) and security requirement (authorization) with the Matchmaker. The security portion of its description is as follows

```
<security:SSH rdf:ID="Capability3" />
<security:KeyProtocol rdf:ID="Requirement3">
    <security:relSecNotation rdf:resource="#Authorization" />
    <security:syntax rdf:resource="#XML" />
</security:KeyProtocol>
<profile:Profile rdf:about="#W1">
    <profile:serviceName>
        <xsd:string>
            <rdf:value>Capability SSH</rdf:value>
        </xsd:string>
    </profile:serviceName>
    <profile:textDescription>
        <xsd:string>
            <rdf:value>This service is cabable of SSH communication.</rdf:value>
```

```
        </xsd:string>
    </profile:textDescription>
    <securityCapability rdf:resource="#Capability3" />
   <securityRequirement rdf:resource="#Requirement3" />
</profile:Profile>
```

The Matchmaker finds a matching service, W2, with the functional description of personal banking and SSH as both its security requirement and capability.

```
<security:SSH rdf:ID="Capability4" />
<security:SSH rdf:ID="Requirement4" />
<profile:Profile rdf:about="#W2">
    <profile:serviceName>
        <xsd:string>
            <rdf:value>Capability+Requirement is SSH</rdf:value>
        </xsd:string>
    </profile:serviceName>
    <profile:textDescription>
        <xsd:string>
            <rdf:value>This service is cabable of SSH communication.</rdf:value>
        </xsd:string>
    </profile:textDescription>
    <securityCapability rdf:resource="#Capability4" />
   <securityRequirement rdf:resource="#Requirement4" />
</profile:Profile>
```

References

[1] B. Atkinson, G. Della-Libera, S. Hada, M. Hondo, P. Hallam-Baker, J. Klein, B. LaMacchia, P. Leach, J. Manferdelli, H. Maruyama, A. Nadalin, N. Nagaratnam, H. Prfullchandra, J. Shewchuk, and D. Simon, 2002. `http://www-106.ibm.com/developerworks/webservices/library/ws-secure/`.

[2] M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon. Xml-signature syntax and processing rules. `http://www.w3.org/TR/2001/PR-xmldsig-core-20010820/`, August 2001.

[3] A. M. L. Committee. Daml+oil. `http://www.daml.org/2001/03/daml+oil.daml`, March 2001. See `http://www.daml.org/committee/` for committee members.

[4] The DARPA Agent Markup Language (DAML). `http://www.daml.org`.

[5] DAML services. `http://www.daml.org/services`.

[6] G. Denker, J. Hobbs, D. Martin, S. Narayanan, and R. Waldinger. Accessing information and services on the DAML-enabled web. In S. Decker, D. Fensel, A. Seth, and S. Staab, editors, *2nd. Intern. Workshop on the Semantic Web SemWeb'2001, Workshop at WWW10, Hongkong, China*, May 2001. `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-40/`.

[7] D. M. (editor), A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara. DAML-S 0.7 draft release. `http://www.daml.org/services/daml-s/0.7/`.

[8] R. Fikes, J. Jenkins, and G. Frank. JTP: A System Architecture and Component Library for Hybrid Reasoning. http://www.ksl.stanford.edu/KSL-Abstracts/KSL-03-01.html, 2003.

[9] J. Hendler. Agents on the web. *IEEE Intelligent Systems, Special Issue on the Semantic Web*, 16(2):30–37, March/April 2001.

[10] P. Lambrix. Description logics references. `http://www.ida.liu.se/labs/iislab/people/patla/DL/references.html`.

[11] S. McIlraith, T. Song, and H. Zeng. Mobilizing the semantic web with DAML-enabled web services. In S. Decker, D. Fensel, A. Seth, and S. Staab, editors, *2nd. Intern. Workshop on the Semantic Web SemWeb'2001, Workshop at WWW10, Hongkong, China*, May 2001. `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-40/`.

[12] S. McIlraith, T. Song, and H. Zeng. Semantic web services. *IEEE Intelligent Systems, Special Issue on the Semantic Web*, 16(2):46–53, March/April 2001.

[13] Ontology inference layer (oil). `http://www.ontoknowledge.org/oil/`.

[14] Simple public key infrastructure (spki). `http://www.ietf.org/html.charters/spki-charter.html`.

[15] K. Sycara, J. Ju, M. Klusch, and S. Widoff. Dynamic service matchmaking among agents in open informatin environments. *ACM SIGMOD Record, Special Issue on the Semantic Interoperability in Global Information Systems*, 1999.

[16] Ietf and w3c xmlsignature working group. `http://www.w3.org/Signature/`.

[17] P. Zimmermann. *The Official PGP User's Guide*. MIT Press, 1995.