Security for DAML Web Services: Annotation and Matchmaking*

Grit Denker[1], Lalana Kagal[2], Tim Finin[2], Massimo Paolucci[3], and Katia Sycara[3]

[1] SRI International, Menlo Park, California, USA
denker@csl.sri.com
[2] University of Maryland Baltimore County, Baltimore, Maryland, USA
{lkagal1, finin}@cs.umbc.edu
[3] Carnegie Mellon University, Pittsburgh, Pennsylvania, USA
{paolucci+, katia}@cs.cmu.edu

Abstract. In the next generation of the Internet semantic annotations will enable software agents to extract and interpret web content more quickly than it is possible with current techniques. The focus of this paper is to develop security annotations for web services that are represented in DAML-S and used by agents. We propose several security-related ontologies that are designed to represent well-known security concepts. These ontologies are used to describe the security requirements and capabilities of web services providers and requesting agents. A reasoning engine decides whether agents and web service have comparable security characteristics. Our prototypical implementation uses the Java Theorem Prover from Stanford, for deciding the degree to which the requirements and capabilities match based on our matching algorithm. The security reasoner is integrated with the Semantic Matchmaker from CMU giving it the ability to provide security brokering between agents and services.

1 Introduction

Today's Internet is a vast information resource. However, its lack of structure and computer understandable 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. expansion of web space. The DARPA Agent Markup Language DAML+OIL [1] is a language that allows the annotation of web pages to indicate their meaning. One of the advantages of DAML+OIL over other markup languages like

* Supported by the Defense Advanced Research Projects Agency through the Air Force Research Laboratory under Contract F30602-00-C-0168 to SRI and contract F30602-00-2-0592 to CMU and DARPA contract F30602-97-1-0215 to UMBC.

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 features allow to capture 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 role in society. 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 of information of the Semantic Web in ways that meet security requirements such as authentication, authorization, and data integrity.

Our work focuses on security aspects for DAML web services. DAML-S [2] is a set of ontologies that support the description of Web services. DAML-S describes a Web service at three levels of abstraction: capability, process and invocation. The *capabilities* of a Web service are expressed by the *Service Profile* which describes high level features of the Web service and, most importantly, input/output transformations produced by the invocation of the Web service. The *process*, expressed by the *Service Model* describes what the service does. Finally, the invocation is described by the *Service Grounding* that describes how to contact the service, for example through asynchronous messaging or remote procedure call, and the format of the information to exchange.

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. Motivating examples are given in Section 2. In Section 3 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. These ontologies are used to describe the security requirements and capabilities of web services and requesting agents. The requirements and capabilities can be specific by stating the particular standards/protocols supported or more generally in terms of the security mechanisms used, the credentials required or notations specified. 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. In Section 4 we propose an algorithm that 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 [3], for deciding the degree to which the requirements and capabilities match based on our matching algorithm. The security reasoner is integrated with the Semantic Matchmaker from CMU [4, 5]

giving it the ability to provide security brokering between agents and services. An extended example explains the working of the system. Section 5 concludes with a brief summary and future work.

1.1 Related Work

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 [6, 7], SAML [8], and WS-Security [9]. Our work aims to provide an layer of abstraction on top of the various existing security-related standards, addressing general security mechanisms (such as confidentiality etc.) without re-defining all the details for specific implementation choices (such as XMLSignature Syntax).

To our best knowledge, no integrating formal framework for this wide array of security-related approaches exists. An overview about security formats by G. Klyne [10] mentions many of the security notions that we formally specified in our ontologies. The openness of the semantic web dictates that there will be no such thing like 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 mechanisms become available, we can extend the existing classes and instances in order to incorporate the latest developments.

Most web based application servers like Apache, Tomcat, and Websphere include basic security functionality including authenticating users (via username/password, certificate, etc.), securing the communication (via https, ssl, etc.), support for sessions, etc. We are modeling these commonly present security features and services so that our ontologies can be used to describe and reason about a wide range of security related concepts.

2 Context and Motivating Example

Our work is targeted toward situations in which agents 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. 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 a matchmaker service (e.g., [4]), 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 for this purpose. 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 the desired characteristics of the requested-service. Along with describing its own security capabilities, it includes requirements for the requested service. The request is sent to a matchmaker who, after finding services that meet the functional requirements of the agent, will utilize the security reasoner to decide the subset of all discovered services that also meet the security requirements of the requester agent.

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 An agent, A, is looking for a travel web service. Agent A, using a matchmaker interface, fills out a template for a requested web service, 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 capable of authenticating itself and communicating in XML. A travel web service, T, registers with the same matchmaker. It provides its name, description, functional capabilities, security requirements and security capabilities. We assume that the travel service requires an agent to be able to perform encryption 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.

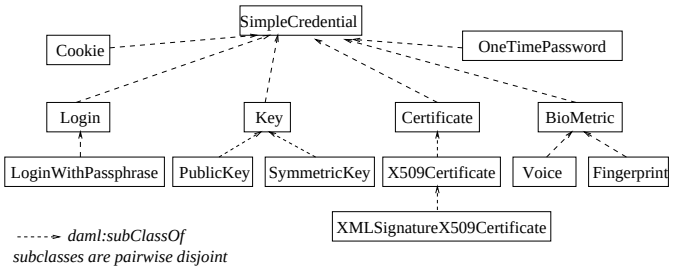

Fig. 1. A credential ontology (class hierarchy I)

Negotiation We do not assume that agents and services always register with their full capabilities. For example, an agent registers with the capability of having some certificate but the web service requires specifically X.509 certificates. 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. Cases like this might be useful when agent or service first want to go through another protocol in which they establish some level of trust before they release what specifics about the credentials or other capabilities they hold. Protocols for negotiation in the context of our work will be subject of future work.

3 DAML Ontologies

Our goal is to define security ontologies in DAML+OIL that allow the annotation of agents and web services with respect to various security related notions such as access control, data integrity and others. These ontologies are the basis for doing automatic subsumption reasoning over security annotations. All ontologies can be found at `www.csl.sri.com/users/denker/daml-sec/`. 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 are one of the various well-known 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.

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.

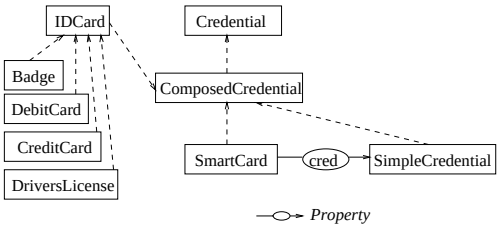

Fig. 2. A credential ontology (class hierarchy II)

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 [7].

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.

Figures 1 and 2 only depict classes and their inheritance relationships. Properties and other restrictions are defined in the ontologies as well. For example the LoginWithPassphrase class has two datatype properties defined, "loginName" and "passphrase", both of type string (see Figure 3). 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 LoginWithPassphrase-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 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 [7]. This way we tie our ontology into an ontology that is being standardized. 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>
```

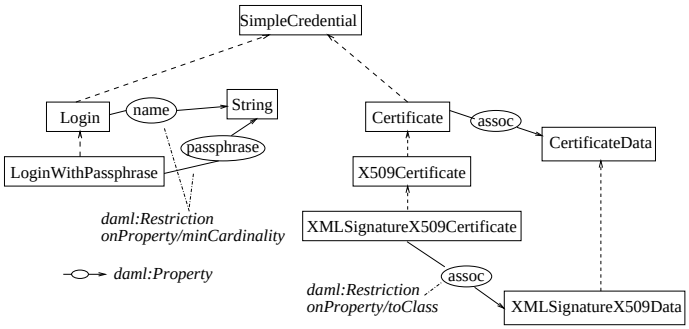

Fig. 3. A credential ontology (some properties)

```
<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>
```

The tie into the standardized specifications is useful to (1) exploit other concepts of widely-distributed specifications or (2) to express more detailed security policies. As an example for (1) one could use the `KeyInfo` element in an XML `Signature` element. `KeyInfo` 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 [7], to PGP public key pairs [11], and SPKI public key pairs [12]. We can exploit those structures by defining specific instance of the class `Credential` in our security ontology. As an example for (2) 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.

3.2 Security Mechanisms

With our ontologies for security mechanism we aim to capture high-level security notations that are commonly used in describing user, agent or service security policies.

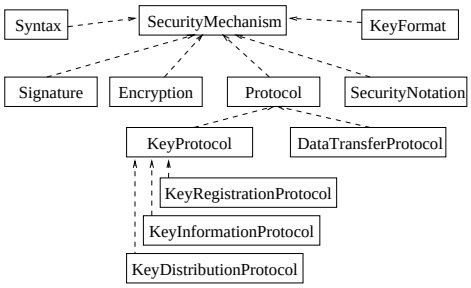

Fig. 4. A security ontology I

We propose an ontology that allows to interface on a high level of abstraction among various security standards and notations. 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. For example, 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. "XML-DSIG" is an instance of the signature class and "OpenPGP-Enc" is of type "Encryption." Specific protocols such as "X-KRSS, X-KISS, Kerberos," or "SOAP" are defined as instances of the appropriate protocol subclasses that satisfy certain restrictions.

Restriction classes are classes in which we constrain the range of one of the object properties "reqCredential, syntax, relSecNotation" etc. Restriction classes are used as "patterns" for certain security notations. For example, an "AuthenticationSubClass" is a class that has as one of its related security notations "Authentication".

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

Thus, if a web service has the authentication class as one of its requirement then it means that the service requires the user to authenticate. How web services can make use of the restriction classes is defined in the next section.

One can define similar restriction classes. For example, one can define a class where one of the required credentials is X.509 or where one of the syntax used is XML. We defined protocols in terms of their characteristics. 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>
```

XKMS is a protocol that provides authentication and key distribution and it uses XML as syntax. For more examples of restriction classes and protocols see `www.csl.sri.com/users/denker/daml-sec/security.daml`.

We choose to define restriction classes to represent security requirements because we want to make use of the inherent features of the description logic (DL), on which DAML+OIL is based, and the tools that are available for DL approaches. Subsumption reasoning is one of the well-defined techniques for description logics we which employ for our purposes. It allows to decide whether security capababilities and requirements are identical or whether they are in an appropriate inheritance relationship. Thus, we spare ourselves to define special-purpose algorithms that could achieve the same for lists of specific capabilities. Our approach takes advantage of the current technology to the best of its ability.

3.3 Merging Security Ontologies and DAML-S

The missing link between our security ontologies and web services is introduced next. We define a SecurityMechanism to be a subclass of ServiceParameter. Then we can declare two new properties for web services, namely "securityRequirement" and "securityCapability" of type SecurityMechanism.

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>

<profile:Profile rdf:ID="RequestServiceProfile1">
  <profile:serviceName> ... </profile:serviceName>
  <profile:textDescription> ... </profile:textDescription>
  <sec-requirement rdf:resource="#Sec2"/>
</profile:Profile>
```

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

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

<profile:Profile rdf:ID="RegisteredServiceProfile1">
  <profile:serviceName> ... </profile:serviceName>
  <profile:textDescription> ... </profile:textDescription>
  <sec-capability rdf:resource="#Sec1"/>
</profile:Profile>
```

The question is, whether the registered service is a match for the request. Here is where our security reasoner comes into play. For the given example, 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 that handles security matching requests is discussed below.

4 Security Reasoner

Consider the example from Section 2. Agent A that is capable of performing Open-PGP encryption and requires a communicating service to be capable of authenticating itself and communicating in XML, 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 to be capable of encryption and itself to be 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. They can be either instances of defined security protocols like XKMS and Open PGP or collections of instantiated characteristics of these protocols like encryption and authentication. 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 implemented 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*. Perfect match is the best match possible and no match is the worst.

4.1 Security Matching Algorithm

In this section we describe the algorithm used to decide whether two security descriptions are related. 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, but instead can 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, 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 authorization as its related security notation, then, as the protocol SSH has authorization as related 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 the capability has the 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: This is the case 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 in case III there is no general match, then it is considered a no match. If for example, the requirement is for authorization and the capability is authentication and key distribution.

4.2 Integration into the Service Matchmaker

The main objective of Agents and Web services is to perform tasks such as providing information about a stock quotes, or Weather patterns, or purchasing and exchanging goods. To this extent, agents have two types of capabilities: a functional capability that describes the tasks performed by the agent and what the agent achieves; and security capabilities that specify constraints on agent and Web services communications. Any Internet wide registry of agents and Web services should provide a matching mechanism that applies on both dimensions: matching functional capabilities allows requesters to locate providers on the bases of what they do, security matching guarantees that the requester and the provider can interact.

The DAML-S/UDDI Matchmaker greatly improves on current web services registries, specifically UDDI. First, it expands UDDI by supporting the representation of capabilities of Web services; in addition, via the ontologies and security reasoning algorithm presented in this paper, it allows the representation of security capabilities and requirements that restrict the ability of two agents or Web services to interact.

To accomplish this task, the security reasoner has been integrated with the DAML-S/UDDI Matchmaker [4, 5] which uses DAML-S to empower the UDDI web services registry [13] with functional capability matching. The DAML-S Matchmaker performs two tasks, the first one is to store advertisements of capabilities expressed as DAML-S profiles, the second one is to locate which advertisement matches the requests of capabilities the matchmaker received from requesting agents. Currently, the matchmaker uses only input and output information of DAML-S profile specifications. Thus, currently the matchmaker would not be able to handle our security specifications if we would specify them as preconditions of the web service. Moreover, the matchmaker does only consider subsumption on classes, whereas in our example (see Section 4.3) it is necessary to also include subsumption reasoning on individuals. For these reasons, we decided to use JTP's subsumption reasoning in connection with our restriction classes. Moreover, we proposed the security matching algorithm in Section 4.1 because we wanted to support a broader range of subsumption decisions, from perfect match, over close match and possibility of negotiation to no-match, and this is not supported in the current Matchmaker implemenation.

When matching advertisements and requests, the DAML-S/UDDI Matchmaker first locates those web services that satisfy the capabilities that the requester expects, second it removes all the web services that do not match the security requirements imposed by the requester. The latter test requires two matches, the first one is that the requester's requirements are satisfied by the provider's capabilities, then that the requirements that the provider are satisfied by the requester. For this purpose, the DAML-S/UDDI Matchmaker calls twice the security reasoning algorithm with the corresponding parameters. Depending on the result of the security reasoning algorithm, the matchmaker accepts a web service as a possible match or not.

4.3 Walk-Through Example

This section is a walk through of two examples of how security annotations of web services and agents are used by our system to provide security specific brokering. The examples demonstrate several features of our security ontology and illustrate 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 service's description

```
<security:XKMS rdf:ID="Capability2" />
<security:KeyProtocol rdf:ID="Requirement2">
    <security:relSecNotation rdf:resource="Encryption" />
</security:KeyProtocol>
<profile:Profile rdf:about="#T">
    ...
    <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.
  - The reasoner tries General Matching, which is the first case of Case II.
  - 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.
  - The reasoner tries the General Matching case of Case II.
  - It locates all the features associated with OpenPGP-Enc, among which encryption is.
  - 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>... </profile:serviceName>
    <profile:textDescription>...</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">
   ...
   <securityCapability rdf:resource="#Capability4" />
   <securityRequirement rdf:resource="#Requirement4" />
</profile:Profile>
```

5 Concluding Remarks

As shown in this paper, the formal definition of ontologies that will enable interoperability of various security frameworks has several advantages. The uniform representation becomes amenable to formal reasoning and it supports agents in selecting appropriate web services. We have presented a security ontology for DAML+OIL that is extensible in that it allows the definition of further security aspects as well as new links to existing standards and frameworks. With the given security framework for DAML+OIL many of the common access control, authentication, and data integrity measures of existing web services can be described. We described our prototype implementation that uses JTP and the DAML-S Matchmaker to enable security matching between DAML+OIL-aware software agents and services.

The current work is restricted to annotation and matchmaking of services wrt security requirements. In the future we will address the connection between a service's profile and its implementation as defined in the grounding.

In the current framework, we focus on some common security notations. These are often embedded into the broader context of trust policies. We plan to extend our work and prototype to allow for security policy specifications. Logics for trust management is another topic of future investigations. There exists already a large body of work on trust management, e.g., [14–16] to name a few. Recent work on frameworks for distributed trust management and policy in multi-agent systems [17, 18] and existing authentication and delegation logics have to be taken into consideration for future logical extensions of DAML+OIL. The first step will be the development of further basic ontologies for deontic concepts (permissions, obligations, and rights), as well as a basic agent and action ontology (requesting a resource, delegating rights, etc) [19]. A theory for reasoning about trust relations will be at the core of decision procedures for web agents that search for reliable information.

Another direction for future work is to extend our framework to composed services. Composition of security features from atomic services is non-trivial and the derivation of security features of a composite service on the basis of the security descriptions of its components will be addressed in future work.

Acknowledgement We thank the anonymous reviewers for their comments that help to improve the paper.

References

1. Committee, A.M.L.: Daml+oil. http://www.daml.org/2001/03/daml+oil.daml (2001) See http://www.daml.org/committee/ for committee members.
2. : DAML services. (http://www.daml.org/services)
3. Fikes, R., Jenkins, J., Frank, G.: JTP: A System Architecture and Component Library for Hybrid Reasoning. http://www.ksl.stanford.edu/KSL-Abstracts/KSL-03-01.html (2003)
4. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.: Semantic matching of web services capabilities. In: ISWC2002. (2002)
5. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.: Importing the semantic web in uddi. In: Proceedings of E-Services and the Semantic Web Workshop. (2002)
6. IETF, Group, W.X.W.: Xml signature. (http://www.w3.org/Signature/)
7. Bartel, M., Boyer, J., Fox, B., LaMacchia, B., Simon, E.: Xml-signature syntax and processing rules. http://www.w3.org/TR/2001/PR-xmldsig-core-20010820/ (2001)
8. (SSTC), O.S.S.T.: Security assertion markup language (SAML). (http://www.oasis-open.org/committees/security/)
9. Atkinson, B., Della-Libera, G., Hada, S., Hondo, M., Hallam-Baker, P., Klein, J., LaMacchia, B., Leach, P., Manferdelli, J., Maruyama, H., Nadalin, A., Nagaratnam, N., Prfullchandra, H., Shewchuk, J., Simon, D.: (2002) http://www-106.ibm.com/developerworks/webservices/library/ws-secure/.
10. Klyne, G.: Framework for security and trust standards. (http://www.ninebynine.org/SWAD-E/Security-formats-20021202.html)
11. Zimmermann, P.: The Official PGP User's Guide. MIT Press (1995)
12. : (Simple public key infrastructure (spki)) http://www.ietf.org/html.charters/spki-charter.html.
13. http://www.uddi.org/pubs/whitepapers.
14. Abadi, M., Burrows, M., Lampson, B., Plotkin, G.: A calculus for access control in distributed systems. ACM Transactions on Programming Languages and Systems **15** (1993) 706–734
15. Blaze, M., Feigenbaum, J., Lacy, J.: Decentralized trust management. In: Proc. 1996 IEEE Symposium on Security and Privacy, IEEE Computer Society (1996) 164–173
16. Li, N., Grosof, B., Feigenbaum, J.: A practically implementable and tractable delegation logic. In: Proc. 2000 IEEE Symposium on Security and Privacy (S&P'00), IEEE Computer Society (2000) 27–42
17. Kagal, L., Finin, T., Joshi, A.: Developing secure agent systems using delegation based trust management. In: Security of Mobile MultiAgent Systems (SEMAS 02) held at Autonomous Agents and MultiAgent Systems (AAMAS 02). (2002)
18. Bradshaw, J., Uszok, A., Jeffers, R., Suri, N., Hayes, P., Burstein, M., Acquisiti, A., Benyo, B., Breedy, M., Carvalho, M., Diller, D., Johnson, M., Kulkarni, S., Lott, J., Sierhuis, M., Hoof, R.V.: Representation and Reasoning for DAML-Based Policy and Domain Services in KAoS and Nomads. In: *Submitted to* AAMAS'03, July 14-18, 2003, Melbourne, Australia. (2003)
19. Kagal, L., Finin, T., Joshi, A.: A Policy Language for A Pervasive Computing Environment. In: IEEE 4th International Workshop on Policies for Distributed Systems and Networks. (2002)