Web Service Semantic Access Control

Margaret Sazio, Miriam A. M. Capretz
Department of Electrical and Computer Engineering
The University of Western Ontario
London, Ontario
Canada, N6A 5B9
{msazio, mcapretz}@uwo.ca

ABSTRACT: *Web services facilitate the sharing of resources between groups within a collaborative environment. Privacy is an important concern when sharing data. Access control can be used to protect privacy by limiting the access privileges to the underlying resources. This paper outlines a framework for applying access control to data providing Web services. The framework makes use of ontologies allowing for a clear description of the functionality offered by the services within the collaborative environment. The implementation of the framework is described using XACML as a basis for the access control. XACML is extended to function with the defined ontologies in order to provide fine-grained access control. Finally, a case study is worked through showing how the proposed framework could be applied within a healthcare environment.*

Keywords: Access Control, Web Service, Collaborative Environment, XACML, Ontologies

Received: 10 May 2013, Revised 18 June 2013, Accepted 25 June 2013

1. Introduction

Web services allow people and organizations to share their resources. Through the use of Web services multiple organizations can agree to share resources in a collaborative environment. Collaborative environments allow a person or organization to share resources with others to work towards common goals [1]. Two important concerns when sharing data are searching and privacy [2]. While Kagal et al [2] are primarily concerned with peer to peer (P2P) networks the same concerns apply in collaborative environments as both situations involve multiple parties sharing information. The searching problem arises from different organizations having their own naming and organizational scheme for their data. When trying to access a certain set of information, it can be difficult when the same piece of information is stored differently in partners' databases within the collaborative environment. Privacy is a concern, as some organizations will not want to expose sensitive information to unauthorized parties. While there is some information that organizations are willing to share, there must be a way to control data distribution based on user privileges. The framework outlined in this paper aims to solve both problems.

In order to deal with privacy concerns, we are using access control to limit what information is shared and with whom. The framework allows for fine-grained access control, allowing for very specific access control policies to be written. The case study included uses role based access control (RBAC) and XACML as a basis for access control with a few additional components. These components aid in role assignment as well as provide a method of filtering out individual data elements.

In order to solve the searching problem we are using ontologies alongside the Web service. This paper describes the addition of access control to a Data Providing Service (DPS) as described by Brown and Capretz [3]. A DPS is a specialized Web service that only allows data retrieval. The DPS is semantically annotated which is defined by a DPS profile. The DPS profile consists of the parameterized view over the domain ontology as well as the bindings between the variables in the view and the syntactical fields of the Web service. A domain ontology is used to model the data. The domain ontology is developed by non-technical stakeholders who are familiar with the subject matter of the DPS. The architecture described in [3] is extended by adding a filtering ontology, developed by non-technical experts, which is then used along with the domain ontology and DPS Profile to define access control policies (ACP). An ACP must contain a target, one or more rules, and a rule-combining algorithm [4]. The target indicates to whom the policy applies; the target of the policy can be a certain role. Each rule must have a target, which is the resource, subject, etc. that the rule applies to, and an effect, which is the result of the rule: i.e. permit or deny. A rule-combining algorithm describes what to do if the rules disagree as to whether the target of the policy should be granted or denied access to a resource. Examples of rule-combining algorithms include deny/permit-override where a single deny or permit decision decides the outcome for the entire policy, or first-applicable where the decision from the first applicable rule in the ACP acts as the decision for the entire policy.

The paper is organized as follows: Section II describes related work, Section III outlines the framework for the solution, Section IV provides implementation details, Section V contains a case study involving the use of the proposed framework in a fictional healthcare setting, and Section VI concludes the paper.

2. Related Work

The use of ontologies with Web services has been previously researched as a way of extending Web services. For example, Zhao et al [5] use domain ontologies in order to determine if attributes are similar. Attributes are sent from the requester to the provider and if the attributes are determined to be similar to attributes that appear in an ACP, access to a resource will be granted or denied based on the rule for a similar resource. The attribute mappings are saved for future use. Similarly, Priebe et al [6] and Hai-Bo [7] also make use of ontology mapping systems for access control. Priebe et al [6] extend the XACML standard by adding an inference engine and an ontology administration point. With these additions, attributes used in the ACP can be inferred from user attributes. The main contribution of Hai-Bo [7] is to add semantics to attribute based access control (ABAC) to create S_ABAC. The ontology management system used for the S_ABAC framework is able to take domain information and infer attributes associated with it. When a user requests permission to access a resource, the Policy Administration Point (PAP) is able to form '*semantic-aware*' access policies based on the information stored in the ontology management system.

Hai-Bo [7] makes use of a Single Sign-On (SSO) system that retrieves user attributes in order to make decisions based on an ACP. In our case we make use of an SSO system that takes user attributes and maps users to roles.

The existing access control solutions make use of ontologies while considering the domain as well as user attributes, however, the use of ontologies does not extend to the actual filtering of the resources. With the addition of a filtering ontology, we are able to facilitate the writing of ACPs as well as allow for fine-grained access control.

3. Framework

The first step in the process of creating access control policies is for non-technical stakeholders to agree on both domain and filtering ontologies. It is assumed, that at this point the DPS has already been developed or that it will be implemented separately from the access control system. In that case, the domain ontology will be formally defined with the implementation of the DPS. Ideally, the addition of access control would not necessitate the modification of the domain ontology as any changes would likely result in changes in the DPS. The domain ontology concentrates on describing the resources and users, whereas the filtering ontology consists of a set of filtering classes. There is a hierarchy to the filtering classes so it is possible to grant or deny access to a parent filtering class and the rule will apply to all children filtering classes. The terms defined in the domain ontology are grouped into filtering classes through a set of Domain to Filtering rules (D2F). Brown et al [8] presents a definition for D2F rules. Any resources not specifically linked to a filtering class are mapped to a general filtering class. This general class is needed especially in the case where the default permission state is to allow access unless a user has been specifically denied such access. Using this general class, a simple access control policy (ACP) can be defined that denies access to every possible resource. The process of creating the DPS does not change with the addition of access control. At the same time or after the basic DPS is being created, the access control component of the DPS can be generated as well. The filtering ontology and D2F rules are combined to form the Published Filtering Description (PFD). The PFD generated this way can then be used during runtime to evaluate requests for resources.

In order to add access control to the DPS, our solution has two main components: an authorization component as well as a decision-making component. The authorization component is responsible for identifying a user and determining the user's role or roles. Since it could end up being very time-consuming for the system maintainer to manually assign a role to each user a Single Sign-On (SSO) system would be used to facilitate the process. Each user would initially be described using attributes that have been pre-agreed on by all organizations within the collaborative environment. Each organization will have their own roles that correspond to certain attributes. The SSO system is responsible for holding all of the role assignment rules for each organization and therefore is be able to assign user roles. The roles are derived based on: the organization that owns the data, the attributes of the user and the type of data that the user wants to access. The user attribute data as well as the role assignment rules stored by the SSO system would remain completely separate from the framework formed by the organizations that own the data. Adding, modifying, adding or removing user attributes or role assignment rules would be the responsibility of the SSO system administrator, based on requests from the organizations involved. Using this method, the user does not need to be known to each organization within the collaborative environment before being considered for access to data. This is beneficial, as each organization may consist of a great number of employees and trying to keep up-to-date records with role assignment for each individual would be a very challenging task. Through the use of different roles, an organization can be precise in assigning access control to each class of data provided by their service. The number of defined roles limits how specific the ACPs will be, as the number of people belonging to each role will change.

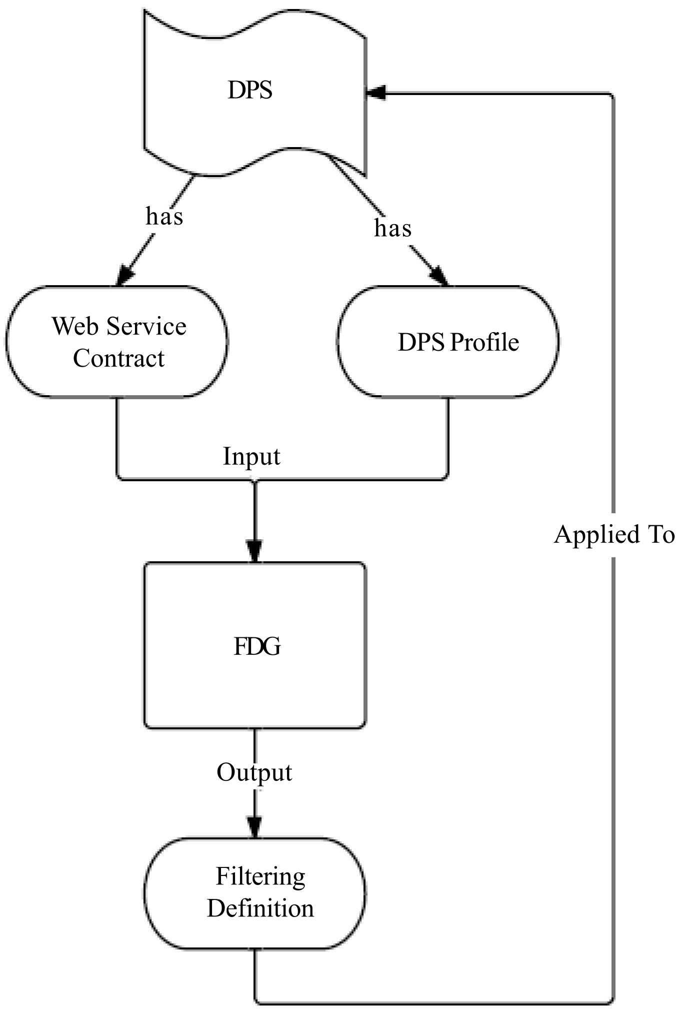

Figure 1. The Filtering Definition Generator (FDG) Process

The second component: the decision-making component, is more complex than the authorization component. The decision-making component includes the filtering definition generator (FDG), the filtering definition, ACPs as well as the filtering class decision point (FCDP). The FDG is only needed when either the DPS Profile or the Web Service contract changes and it is responsible for re-creating the filtering definition. The FDG takes the service contract as well as the DPS Profile as inputs. Using these inputs, a filtering definition is created that takes into account the domain ontology and ensures that the schema will be preserved. The filtering definition is the part of the decision-making component that performs the filtering of the requested resource. Once the decision to share the resource has been made and the resource has been tagged with the appropriate decision label, the filtering definition removes the resources that have been denied, so they are not shared with the user. Figure 1 illustrates how the filtering definition is generated by the FDG using the DPS profile and Web service contract, extracting the required ontology terms and definitions.

The ACPs are defined based on the filtering ontology (a set of filtering classes). Each organization within the collaborative environment generates its own ACPs where the target of the policy is one or more of that organizations defined roles and the target of individual rules contained in the policy are the classes defined in the filtering ontology. There may be additional conditions, such as time or location constraints contained within the ACP. These constraints could be added in cases where a user may have access to a particular resource only if the user is trying to get access during work hours, at the user's work computer. As the ACPs are defined using the filtering ontology, in order for them to be understood, we need the FCDP. The FCDP is the component that determines which filtering class a requested resource belongs to, and it operates on per-request basis with the possibility of using a caching method. Through the use of the FCDP, we can limit the number of times an ACP must be checked for a given request, since the filtering class may contain several elements. By retrieving the filtering class of an element before requesting permission to access it from and ACP, at most the ACP will be accessed as many times as there are filtering classes. As we are implementing fine-grained access control, each individual element of a resource will be checked to see if access should be granted or denied to a user. Some of these elements may have filtering classes in common. The FCDP will be used to determine which class an element belongs to before it is compared to the ACP. When elements within a resource share a filtering class, the comparison against ACP only needs to occur once for that class, rather than being compared for every single element. The FCDP would check the filtering class with little effort, therefore, speeding up the decision making process.

4. Implementation

As a Data Providing Web Service (DPWS) is a specialized DPS the implementation uses standard Web technologies. The eXtensible Markup Language (XML) [9] is used for the underlying data structures and as such other XML based technologies are used for the DPS. The Web Services Description Language (WSDL) [10] is used to describe the Web service contract and Simple Object Access Protocol (SOAP)[11] is used for exchanging information.

The filtering definition is implemented as an eXentensible Stylesheet Language Transformation (XSLT) [12]. A resource that can be requested will be represented using XML. The filtering definition is applied on the resource XML after the XML has been labeled with the permit/deny decisions in order to filter out the elements that have been denied while ensuring that the schema remains intact. The filtering definition is created once, after the Web Service contract and DPWS profile have been finalized. As long as these artifacts do not change, the FDG will not be needed. The FDG creates the filtering definition by building permit and deny options for every XML element. The permit options will include the element data while the tags will reflect the domain ontology. The deny option will either involve the element being omitted from the document or having the element included with "*Deny*" as the data. Which deny option applies depends on whether or not the element is required to maintain the integrity of the XML schema. The term "*Deny*" is added as the data so that the user that has made the request can clearly see that this piece of information was denied and not that it was simply missing from the data set.

The implementation of the solution uses the eXtensible Access Control Markup Language (XACML)[4] as a basis for access control. Components have been added to the basic XACML architecture to allow for fine-grained filtering based on the pre-defined domain and filtering ontologies. The basic XACML components are the Policy Decision Point (PDP), which compares a resource request to the access control policy (ACP) and determines whether resource access should be allowed or denied. In our case, the PDP is making decisions based on an assigned role as well as the filtering class that a resource belongs to. The Policy Information Point (PIP) has access to information about the subject of an ACP, which we are referring to as the '*user*'. We assume that the role of the user must always be determined before attempting to make a decision, thus the PIP is involved in every request. The PIP implements the authorization component that handles user sign-in and role assignment. The Context Handler is what connects the various components by formatting requests and responses in a way that they can be understood by wherever the request goes next. The Context Handler in this case has the extra job of labeling the elements of the XML based on the decision from the PDP. It also caches results from the PDP as to the decision for a given filtering class so that duplicate requests are not made in the event that the XML snippet contains elements belonging to the same filtering class.

The authorization component has access to information about a set of attributes that describe a user. The authorization component also has access to roles assignment rules, based on attributes for each organization. After a user has signed-in to the system using a unique identifier, in this case, a unique username, it is possible for a user to request access to information. Using attributes association with the user, as well as the role assignment rules put forth by the organization that owns the requested data, a role will be assigned. Depending on what data the user wants to access, a user may be assigned multiple roles during a single session where a user's role at any given time will correspond to the owner of the current data requested. In some cases, it may be beneficial to allow for a user to '*sign-in*' without having a unique identifier. This would lead to a user that has no values associated with the required attributes and therefore, such a user will not belong to any role. In the case where the general public would be allowed access to limited data, a user without a role could still be granted some access. As each organization can set their own rules for which attributes define each role, a different role will be set for the user for each organization. These roles are saved until the user sign out of the system, which limits the amount of overhead in making a request. This could be improved by multiple organizations using the same rules to define the attribute to role relations as the user would then be identified by a single role, however, this is not always possible in a collaborative environment due to a user having different levels of access in each of the different organizations.

The components we are adding to the basic XACML architecture are the filtering definition (FD) and filtering class decision point (FCDP). The FCDP uses the PFD, which contains the D2F rules, and DPWS Profile in order to determine what filtering class an individual XML element belongs to. It takes an XML described in local terms and finds the mapping to the domain ontology. It then checks all D2F rules to figure out which filtering class the element belongs to. The FD is an XSLT that will be used with the labeled XML from the Context Handler. It translates the XML so that each element in the output is referred to in domain ontology terms as well as removes elements that have been labeled "*Deny*". In the case where removing an element would render the XML invalid with regards to the schema, the XSLT keeps the element tag and assigns the value "*Deny*". The process of user request following a successful sign-in is shown in Figure 2.

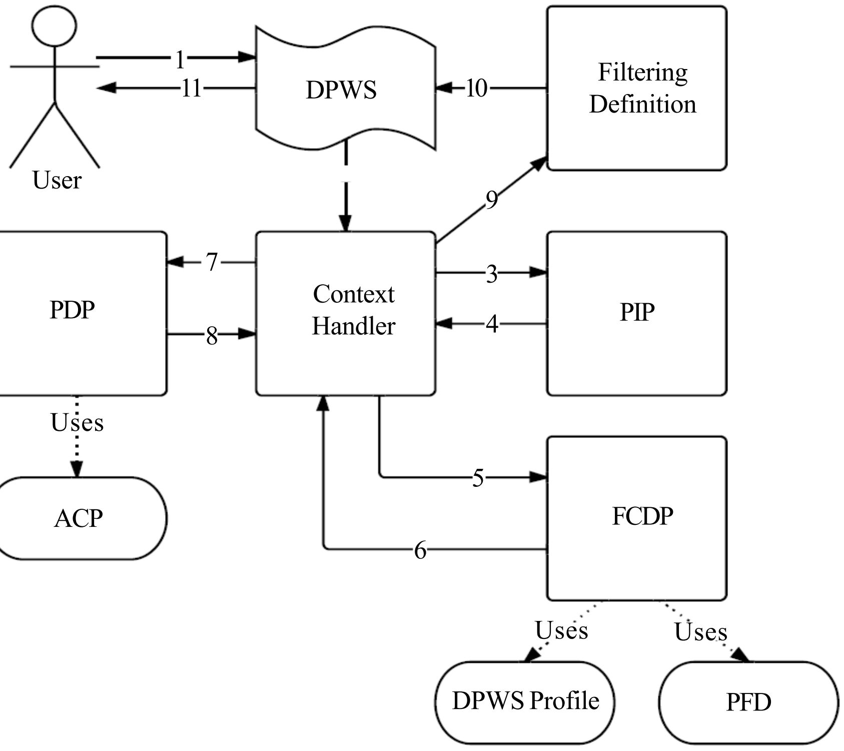

Figure 2. Request/Response Process

Step 0: A user signs in through the SSO system using a username and password combination. (Using XACML elements, the SSO system is shown as the PIP)

Step 1: A user requests a data resource from the DPWS.

Step 2: The DPWS forwards the request to the Context Handler where the request will be formatted in such a way that the PDP can interpret it.

Step 3: The Context Handler at this point only has knowledge about the resource itself and is lacking information about the user. As such, it will request information about the user from the PIP. At this point, the PIP will determine the user's role based on the attributes associated with the unique username and the role assignment rules outlined by the owner of the data resource requested.

Step 4: The PIP will return to the Context Handler the role name corresponding to the user. This role name is needed as part of the request to the PDP.

Step 5: The Context Handler will send a single XML element to the FCDP.

Step 6: The FCDP will use the PFD and the DPWS profile in order to determine which filtering class that element belongs to. The filtering class will be returned to the Context Handler where it is needed to complete the PDP request.

Step 7: The Context Handler sends the resource request to the PDP.

Step 8: The PDP uses the XACML ACP to determine if the user should be granted permission to that filtering class. The result will be returned to the Context Handler where the corresponding element may be labeled as '*Deny*' based on the decision. We are able to label only the denied resources, as our default action in this case is to permit access to any resource unless it has been explicitly denied. Any element that does not receive a label will be permitted without the need for any label comparison during the filtering process.

Steps 5 through 8 are repeated for each element in the XML that has been requested by the user. As the PDP is basing its decisions on the filtering class rather than on a specific element, if the Context Handler, through the use of the FCDP, determines that the XML elements belong to the same filtering class, the elements will all be labeled accordingly. This way, we limit the number of requests that must be evaluated by the PDP.

Step 9: After each XML element has been evaluated and labeled based on the PDP's decisions, the entire document will be sent to the filtering definition where it will be filtered.

Step 10: The filtering definition is an XSLT which will be applied against the XML. The labeled XML is sent to the filtering definition where each element that has been labeled '*Deny*' will be removed. In some cases, the filtering definition will determine that the element cannot be removed completely as it would break the schema. In these cases, the element tag would remain in the output although if a value is needed, that value would be '*Deny*'.

Step 11: The filtered response is returned to the user.

5. Case Study

The case study is based on a health care collaborative environment. There are many different hospitals, clinics, etc. all requiring to share information with each other while controlling exactly what information to disclose. In this case, we will assume that the DPWS is being developed concurrently with the access control component, although aside from the initial step completed by non-technical stakeholders, the implementation of the DPWS does not interact with the implementation of the access control component. In order for the data to be filtered, the first step is for non-technical stakeholders to come up with both domain ontology and filtering ontology. In our case study the domain and filtering ontologies are shown in figures 3 and 4.

Figure 3 demonstrates two classes that are part of the domain ontology: physician and patient. Each of those classes has its own contact information, which consists of multiple attributes. Figure 4 shows the example filtering ontology consisting of four filtering classes. The PII (personally identifiable information) class is the parent of both the PatientPII and PhysicianPII classes. The General class is parent to all filtering classes and contains all resources that would not have otherwise been sorted into other filtering classes. Rules belonging to an ACP would have a filtering class as the target, allowing for permission to be granted or denied based on the class as a whole. Based on these ontologies, we can come up with the following domain to

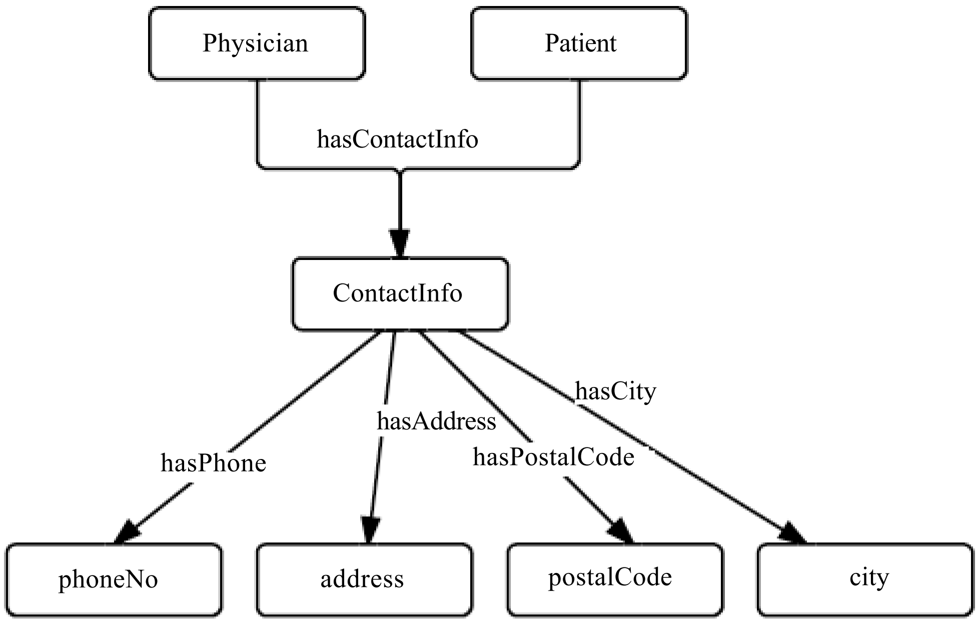

Figure 3. Domain Ontology

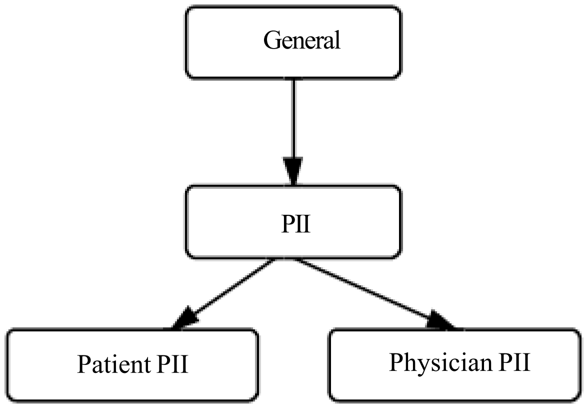

Figure 4. Filtering Ontology

filtering relation (D2F):

```
SET ?contactInfo, ?phone, ?address, ?city AS
                          <http://example/PhysicianPII>
                             WHERE {
                             ?p rdf:type Physician.
                              ?p hasContactInfo ?contactInfo.
                              ?contactInfo rdf:type ContactInfo.
                            OPTIONAL {
                                   ?contactInfo hasPhoneNo ?phone.
                                  ?phone rdf:type PhoneNo.
                                  ?contactInfo hasAddress ?address.
                                 ?address rdf:type Address.
?contactInfo hasCityName ?city.
        ?city rdf:type CityName.}
} EXCEPT { [?contactInfo; hasPostalCode] }
```

The D2F maps contactInfo as well as optional elements phone, address and city to the filtering class PhysicianPII. The Except statement indicates that while contactInfo is part of PhysicianPII and contactInfo has an element postalCode, the postalCode element does not belong to the PhysicianPII filtering class. As postalCode is not part of any other filtering class, thus it will be assigned to the '*General*' class.

In this collaborative environment, there are four organizations: '*A*', '*B*', '*C*', and '*D*'. For the role assignment, the role mapping for organization '*C*' is be given by the following:

❖ $(Job\ description = researcher) \wedge (Employer = A \vee B \vee D) \rightarrow Role = External\ Researcher$

❖ $(Job\ description = researcher) \wedge (Employer = C) \rightarrow Role = Researcher$

❖ $(Job\ description = any) \wedge (Employer\ != A\ \vee B\ \vee C \vee D) \rightarrow Role = General\ Public$

The ACPs would then be written using the 3 defined roles: External Researcher, Researcher, or General Public. With the use of a *General Public* class, a user would be able to sign-in to the system as a guest user. This guest user would have no *Job description* or *Employer* and therefore be assigned to the *General Public* role.

After the preliminaries have been done, the DPWS along with the corresponding access control extension will be published, where users within the collaborative environment can access it. For the purposes of this case study, a user working as a '*researcher*' for organization '*A*' is requesting access to physician information.

Step 0: The user must sign-in to the system. The sign-on authority uses a username and password combination to authenticate the user.

Step 1: The user requests physician information. The following is the result of that request assuming that the user has full access to the detailed information.

```
<Physician>
        <physicianID>123456789</physicianID>
        <Name>Jane Example</Name>
         <Contact>
                <address>111 Address Road</address>
                <city>London</city>
                 <postalCode>M1M2M2</postalCode>
                  <phone>5194224242</phone>
         </Contact>
</Physician>
```

Step 2: The context handler attempts to format the request for the PDP but it is still missing some information about the user's role and about the filtering class of the current XML element.

Step 3: The Context Handler requests the user's role from the PIP. At this point the PIP (SSO system) will look at the user information and see that the user's job description is 'Researcher and is employed by organization '*A*'. These two attributes map the user to the role '*External Researcher*' with respect to organization '*C*' who owns the desired resource.

Step 4: The PIP returns the role '*External Researcher*' to the context handler where this information is added to the PDP request.

Step 5: The context handler takes the first element <physicianID> and sends a request to the FCPD to determine the filtering class.

Step 6: The element <physicianID> has not been explicitly given a filtering class which will result in it being classified as '*General*'.

Step 7: The context handler sends a request to the PDP asking if an '*External Researcher*' has access to 'General'.

Step 8: The PDP will compare ACPs with the request. In this case, an '*External Researcher*' is allowed access to 'General' data and as such, a '*Permit*' response will be returned to the context handler. As only 'Deny' responses will be labeled, the context handler is now done with this element and can move on, returning to step 5.

Return to step 5 for the next element.

Step 5: The next element is <Name> and a request is sent to the FCDP asking for a filtering class.

Step 6: The <Name> element belongs to the same '*General*' class as <physicianID> as it was also not explicitly mapped to a filtering class in the D2F relations. As the context handler has already made an identical request to the PDP regarding access to the '*General*' class, no subsequent request is made and the <physicianID> element is left unlabeled as access is permitted.

Return to step 5 for the next element.

Step 5: The context handler takes the next element, <Contact>, from Physician and asks the FCDP what class it belongs to.

Step 6: The FCDP determines that the term '*Contact*' maps to '*ContactInfo*' and that '*ContactInfo*' is a member of the PhysicianPII filtering class. However, since Contact is made up of sub-elements and it is possible for the sub-elements to belong to different filtering classes, Contact itself will not be labeled as PhysicianPII and instead, each element will be checked to determine its filtering class.

Return to step 5 for the next element.

Step 5: The next element, <address> is sent to the FCDP.

Step 6: The FCDP determines that <address> is a member of the '*PhysicianPII*' filtering class.

Step 7: The context handler has not yet made a request to the PDP regarding access to the '*PhysicianPII*' class and therefore sends a request to the PDP asking if an '*External Researcher*' has access to '*PhysicianPII*'.

Step 8: The PDP compares the request to the ACP and determines that the '*External Researcher*' should not have access to '*PhysicianPII*'. The decision is sent to the context handler where the Contact element is labeled "*Deny*".

Steps 5-8 are repeated for every element in the XML request.

Other elements of Contact: city and phone are all determined to be PhysicianPII by the FCPD and are therefore labeled "*Deny*" since we know the decision without having to ask the PDP again. The postalCode element does not belong to PhysicianPII and instead belongs to the filtering class '*General*'. Based on the previous results, '*External Researcher*' is allowed access to postalCode.

The labeled XML will look like the following:

```
<Physician>
<physicianID>123456789</physicianID>
        <Name>Jane Example</Name>
        <Contact>
<address permission = "Deny">111 Address Road</address>
        <city permission = "Deny">London</city>
        <postalCode>M1M2M2</postalCode>
        <phone permission = "Deny">519-422-4242</phone>
        </Contact>
</Physician>
```

Step 9: The labeled XML is then run against the filtering definition. Part of the filtering definition is shown below. The service contract indicated that the ContactInfo element is a required element for the schema, although the ContactInfo elements are not required. Using the DPWS, the filtering definition also includes the mapping from the local XML format to the domain ontology.

```
<?xml version = “1.0" encoding = “UTF-8"?>
<xsl:stylesheet xmlns:xsl = “http://www.w3.org/1999/XSL/Transform” version = “1.0">
<xsl:output method = “xml” indent = “yes”/>
      …
<xsl:template match = “Contact”>
    <xsl:choose>
       <xsl:when test = “@permission = “Deny’”>
           <ContactInfo/>
       </xsl:when>
       <xsl:otherwise>
          <ContactInfo>
             <xsl:choose>
                  <xsl:when test = “./address[@permission = ‘Deny’]”/>
                     <xsl:otherwise>
                         <address>
                             <xsl:value-of select = “address”/>
                         </address>
                     </xsl:otherwise>
             </xsl:choose>
                    …
           </ContactInfo>
       </xsl:otherwise>
     </xsl:choose>
  </xsl:template>
      …
</xsl:stylesheet>
```

Step 10: After the labeled XML is sent to the filtering definition, the output is as follows:

```
Filtered Response

<Physician>
   <physicianID>123456789</physicianID>
   <Name>Jane Example</Name>
    <Contact>
        <postalCode>M1M2M2</postalCode>
    </Contact>
</Physician>
```

Step 11: The filtered response will be returned to the user that made the request.

The user would then be free to make more requests to the DPWS without the need to sign-in again. In the event that the user requests more data resources from the same source (in this case organization ‘*C*’) the request process would proceed following the same steps outlined above except for Step 3. As the PIP has already determined the user’s role based on the rules outlined by organization ‘*C*’, subsequent requests to the PIP are not necessary, as the user’s attributes have not changed. In the event that the user would want access to resources owned by organizations ‘*A*’, ‘*B*’, or ‘*D*’, another request to the PIP would be made where the user would be assigned a different role.

6. Conclusion and Future Work

In conclusion, this paper provides a solution for fine-grained access control for collaborative environments. Through the use of domain and filtering ontologies, we are able to build a DPWS that can be used and understood by a variety of organizations wanting to work together and share data. The proposed framework allows for dynamic role assignment based on a set of attributes. The use of a SSO allows users to sign in once and be granted access to all the data available in multiple DPWSs, while at the same time provides a way to gather attribute information that is used in role assignment. . Implementing the sign-in system as such allows for a reduced workload in the initial setting up the system as individual users do not need to be manually assigned roles. The dynamic role assignment also makes it easier for users to change roles as needed as assignment is based on attributes which can be changed by the user's local environment. In order for the SSO system to be effective, trust must be present within the organizations forming the collaborative environment, as it is the users' local organization that designates user attribute values. This drawback is dealt with by having each organization have its own role assignment algorithm. In this way, access can be limited when an external user is trying to access data and users from different organizations can be placed in different roles where the users from a more trusted organization are granted broader access to the data.

The filtering classes are what allows the access control to be fine-grained. By classifying individual data elements into filtering classes, ACPs can be written that are very specific as to what they grant or deny access to without the need for individual elements to be listed. The way the filtering elements are incorporated with a basic XACML structure allows us filter individual data elements using the filtering classes without adding of extra overhead when checking ACPs. By having the filtering classes checked for each element before comparing against the ACP, we can limit how many times the ACP is checked while still allowing for fine-grained access control.

Future work includes testing the solution with a larger, more complex case study to test the results in a more realistic environment. The framework will then be extended to work in a cloud environment.

References

[1] William Tolone, Gail-Joon Ahn, Tanusree Pai, Seng-Phil Hong. (2005). Access control in collaborative systems. *ACM Comput. Surv.* 37 (1) 29-41, March.

[2] Lalana Kagal, Tim Finin, Anupam Joshi. (2003). A Policy Based Approach to Security for the Semantic Web, *In*: The Semantic Web - ISWC, Sanibel Island, FL, p. 402-418

[3] Brown K-P, Capretz M-A-M. ODEP-DPS. (2013). Ontology-Driven Engineering Process for the Collaborative Development of Semantic Data Providing Services. *Information and Software Technology*. DOI: 10.1016/j.infsof.2013.02.011.

[4] OASIS. (2005, February) eXtensible Access Control Markup Language (XACML) Version 2.0. [Online]. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

[5] Yi Zhao, Zhong Li, Wolfgang Halang. (2010). Web service access control with semantic interoperation, *Intelligent Systems and Knowledge Engineering (ISKE), International Conference on*, 426 (429)15-16, Nov.

[6] Priebe, T., Dobmeier, W., Kamprath, N. (2006). Supporting attribute-based access control with ontologies, *Availability, Reliability and Security. ARES. The First International Conference on*, 8, p. 20-22, April.

[7] Shen Hai-Bo. (2010). A Semantic- and Attribute-Based Framework for Web Services Access Control, *Intelligent Systems and Applications* (*ISA*), 2nd *International Workshop on*, p. 1, 4, 22-23, May.

[8] Brown, K. P., Hayes, M. A., Allison, D. S., Capretz, L. F., Mann, R. (2012). Fine-Grained Filtering of Data Providing Web Services with XACML, *Enabling Technologies: Infrastructure for Collaborative Enterprises* (*WETICE*), *IEEE* 21st *International Workshop on*, p. 438,443, 25-27, June.

[9] World Wide Web Consortium. (2003) Extensible Markup Language (XML). [Online] http://www.w3.org/XML/

[10] World Wide Web Consortium. (2001) Web Services Description Language (WSDL) 1.1. [Online]. http://www.w3.org/TR/wsdl.

[11] World Wide Web Consortium. (2000, May) Simple Object Access Protocol (SOAP) 1.1. [Online]. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[12] World Wide Web Consortium. (1999, November) XSL Transformations (XSLT). [Online]. http://www.w3.org/TR/xslt.