Mediating Semantic Web Service Access using the Semantic Firewall

Mariusz Jacyno, Terry Payne
University of Southampton, UK
{mj04r,trp}@ecs.soton.ac.uk

E. Rowland Watkins, Steve J. Taylor,
Mike Surridge,
IT Innovation, Southampton, UK
{erw,sjt,ms}@it-innovation.soton.ac.uk

Abstract

As the technical infrastructure to support Grid environments matures, attention should focus on providing dynamic access to services, whilst ensuring such access is appropriately monitored and secured. Access policies may be dynamic, whereby intra-organisational workflows define local knowledge that could be used to establish appropriate credentials necessary to access the desired service. We describe a typical Grid-based scenario that requires local semantic workflows that establish the appropriate security access, whilst global workflows define how external services are accessed. We present the Semantic Firewall, and the use of Process-based Access Control (PBAC) to mediate service access, and present OWL-S extensions that support additional PBAC access policies. Finally, a prototype implementation that validates this approach is presented.

1 Introduction

The Grid Computing paradigm [10] aims to facilitate access to a variety of different computing and data resources distributed across geographical and organisational boundaries; thus enabling users to achieve (typically) complex and computationally intensive tasks. To realise this vision, much of the research and development in recent years has focussed on directing Grid environments towards establishing the fundamentals of the *technical infrastructure* required [11, 22, 7], and addressing the pragmatic issues of composing, integrating, and utilising services offered by a plethora of independent providers [16, 24, 5].

However, while such a technical infrastructure is necessary to provide an effective platform to support robust communication, interoperation, and service utilisation, issues of security and access policies need to be addressed before we can achieve the goal of secure, dynamically composed and provisioned service-execution. In particular, whilst low-level security concerns (including encryption, authentication, etc) have been addressed, the problems of describing authorised workflows (consisting of the execution of several disparate services) and the policies that are associated with service-access have to date been largely ignored.

Many machine-readable specifications have attempted to address the problem of secure access for Service-Oriented Computing (SOC), and specifically for Web Services, including WS-Security[1], Role-Based Access Control (RBAC), and more recently Process-Based Access Control (PBAC) [20, 21]. Several draft proposals for machine-readable policies have also been proposed, including WS-Policy[2] and WS-SecurityPolicy[3], which both provide declarative languages for defining policies for message security. Coupled with other draft specifications such as WS-Trust[4] and WS-Federation[5], a foundation for next generation Service-Oriented Architecture (SOA) authentication and access control is now emerging.

The enforcement of network security policies between different organisations has long been challenging, and is difficult enough when supporting well defined applications and services, such as web servers, telnet, and ftp servers. However, this becomes even more challenging in the presence of dynamically changing and unpredictable Grid communication needs, as the diversity, availability and reliability of services provided by an organisation (e.g. an e-Science Laboratory) continually changes and evolves; and thus places ever more demands on Network Administrators. Although emerging WS-security standards can be used to manage security and access control, they fail to specify the

[1]http://www.oasis-open.org/committees/wss/
[2]http://specs.xmlsoap.org/ws/2004/09/policy/
[3]http://specs.xmlsoap.org/ws/2005/07/securitypolicy/
[4]http://specs.xmlsoap.org/ws/2005/02/trust/
[5]http://schemas.xmlsoap.org/ws/2003/07/secext/

legitimate steps a client may have to take in order to gain the necessary rights to execute an operation. For example, within a business environment, the procurement of new equipment through an equipment supplier may require the existence of a business account, and present the necessary tokens in order to make a purchase. Whilst the workflow for interacting with the equipment supplier may be defined, access to this service is predicated on the user possessing the necessary security tokens, and thus an additional workflow is needed to define how the user gains these tokens.

Our notion of Process-based Access Control (PBAC) [20, 21] regulates access to web services by considering *resource state* and *user role*. The resource state is represented by a finite state machine, which defines what operations are possible depending on the state of a service. This contrasts with the Web-Service Resource Framework (WSRF) [6] definition of "state"[6]. User roles determine what actions may be permitted based on the "role" a user is playing, rather than just the identity of a user. For example, administration services may only be accessible when a user changes their role from "Scientist" to "Principal Investigator". By combining the user role and the current resource state, PBAC can provide a more flexible and expressive framework for defining security policies.

Although PBAC-protected services typically utilise the same standards as normal Web Services to define their operations and orchestrations, etc., a client invoking such a service will only be successful if the client satisfies the PBAC security constraints. Thus, these constraints need to be defined in a machine-processable manner. Although declarative languages such as BPEL4WS[7] provide a natural basis for defining *resource state* (as a finite state machine), semantically-enabled workflow frameworks (such as OWL-S [2] or WSMO [14]) provide a more natural basis to integrate definitions of *user role* and the associated knowledge that relates such roles within an organisation.

In this paper, we present the *Semantic Firewall*, that utilises extensions to OWL-S [2] to describe PBAC policies and workflows, and demonstrate by means of a use case how such PBAC policies are defined. The novelty of this approach is that the goal achievement may involve more than one actor. This is certainly true in real world service-oriented architectires, and Business-to-Business (B2B) environments. The Semantic Firewall workflow not only facilitates the specification of the goal that should be achieved, but also the type of actor the client should contact in order to achieve it. The specification of different actors is useful in cases where the client cannot satisfy all the security constraints, and thus requires assistance (and consequently interaction) with a third party.

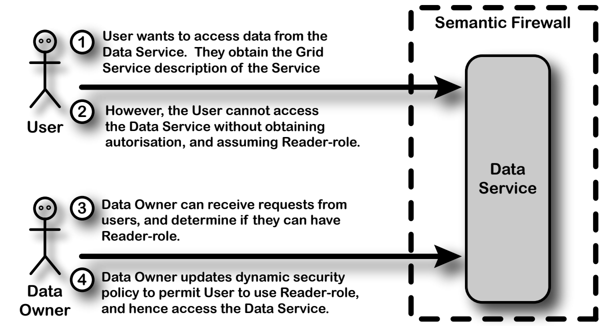


Figure 1. Data Service Use Case Scenario

The paper is structured as follows: Section 2 describes the motivating Use Case study, based on accessing a third-party Data Service. The extensions to OWL-S are presented in Section 3, whereas Section 4 presents a prototype implementation of the Semantic Firewall, its design and implementation. Section 5 gives an overview of the challenges faced and lessons learnt. We discuss related work in Section 6 and conclude in Section 7.

2 Use Case

To motivate the rationale for the Semantic Firewall and PBAC for defining semantically-annotated access policies, we present a use-case based on secure data-storage and data-access delegation. This type of task is typical within inter-enterprise business Grids, such as those created by the GRIA[8] [21] and GEMSS[9] [4] projects. These Grids are typically forced to use some dynamic policy elements to handle changing business relationships, and in some cases, legal constraints over the movement of data [12].

An independent service provider advertises a data service that provides a limited set of operations for manipulating the data it manages. In this scenario, we assume that there are two client-side actors; the *User* and *Data Owner*, each of which have different access control roles, *reader-role* and *owner-role* respectively. Users with the *reader-role* can perform the `read()` operation and access data if it exists. The owner role can also perform the `read()` operation and grant other users read access to data within the data store through the operation `enableRead()`.

The User and Data Owner belong to the same organisation; however this is different to the organisation maintaining the Data Service. Such Data Services are regularly used within Grid workflows to store data large volumes of data generated between different services. By maintaining independent Data Services that can store or "stage" data that is flowing between Grid Services, the need to continually

[6]WSRF describes the state of the application the service represents, rather than explicitly defining the access control in terms of states.

[7]http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

[8]GRIA is now an open source Grid middleware, currently at v4.3, obtainable via http://www.gria.org

[9]http://www.gemss.de

transfer potentially huge amounts of data back and forth between different Users and Grid Services is eliminated.

Data held by the Data Service is divided into *data stagers*, represented using resource identifiers. When a user executes an operation, they specify the data stager which they wish to access, as part of the *context* which is included in the SOAP message header. The PBAC module within the Semantic Firewall extracts this context, along with the user's identity, and verifies whether or not they have access rights to these data stagers.

Figure 1 illustrates this Use Case, by presenting the User (assuming a *reader-role*) and Data-Owner (assuming the *owner-role*). The User can obtain Web Service descriptions (WSDL, etc) which define how to access the Data Service. However, unless the Semantic Firewall's security policy is modified to allow the User access to the data when it assumes the *reader-role*, no access is permitted. The interactions between the User, Data-Owner and Data Service are illustrated in Figure 2. In order to permit the User to read the data from the data service, the User must first obtain read-access by contacting the Data-Owner and requesting the access to the role *reader-role*. To achieve this, the Data-Owner submits a `enableRead()` service request on behalf of the user.

If a User is to successfully invoke the discovered data service, then the User needs to know the process required to acquire the *reader-role* at the Data Service. The workflow published by the Data Service simply defines the operations and orchestration required to access its data (assuming that the User can satisfy its PBAC security policy). However, different organisations may adopt different workflows for granting users access to secured data service; likewise, these workflows may be security sensitive, and hence are only available within an organisation. Thus, the User needs to be able to obtain this workflow to contact the Data-Owner, obtain the *reader-role*, and thus satisfy their goal (i.e. performing the operation `read(dataStagerID)`). Contacting the Data-Owner to request the operation `enableRead(dataStagerID, User)` would therefore become a subgoal. As we can see, service discovery is only the beginning when we consider security constraints on web services.

In the remainder of the paper we will discuss the design and implementation of the Semantic Firewall, a mechanism that can provide answers to how to access a secure web service and possible ways to resolve access failures should these requirements not be met.

3 Extending OWL-S to support PBAC Policy model

To support the definition of PBAC policies that can be used by the Semantic Firewall (SFW), the OWL-S [2] Se-

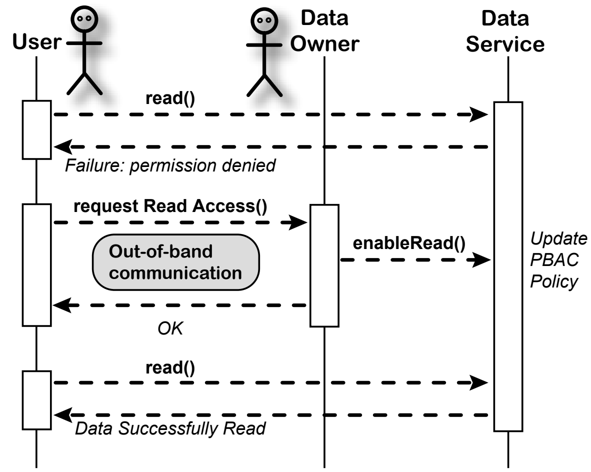


Figure 2. Sequence Diagram of User attempting to invoke Data Service

mantic Web Service ontologies have been extended. As described earlier, a mechanism was required that could support the semantic annotation of service workflows that represent a finite state machine. Thus, not only would OWL-S support the interpretation of hither-to unseen service descriptions, but it would also support the integration and interoperability between publicly available workflows (defining access to the services themselves) and internal policy workflows (that Users would need in order to establish the appropriate credentials to access the services).

OWL-S [2] consists of a set of ontologies designed for semantically describing, choreographing and invoking services and workflows within open, distributed systems. OWL-S provides four high level ontologies, which can be employed, or subclassed, to facilitate the modelling of service descriptions. The *Service model* represents the service itself, and presents three different views on the service; the *Profile model*, which describes what the service does (in terms of a capability description); the *Process model*, which describe how the service works (in term of a process workflow), and a *Grounding model*, which maps the process workflow to a WSDL description of the service.

The capability descriptions of the service and its processes (or operations) are presented in terms of exchanged data resources (i.e. *inputs* and *outputs*) corresponding to the data flow between components, and the state-based notions (i.e. *preconditions* and *effects*) which are used for the logical control of the workflow. In addition, these state-based notions can also be used to represent concepts within the real world (i.e. not representable using data resources). A control and dataflow model is represented as a hierarchical workflow, using one of several workflow constructs (e.g. sequence, choice, split-join operators, etc).

The mapping of PBAC policy model into OWL-S representation required us to focus on an effective way of defin-

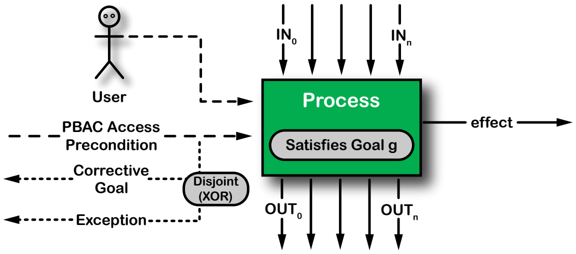


Figure 3. Extending the OWL-S *Process* class with PBAC preconditions, corrective goals, and exceptions

ing operations users could execute on a service in order to achieve their goals. Because these operations have associated security constraints, we found OWL-S processes together with their preconditions and effects relevant while tackling the representation problem.

PBAC policies extend the OWL-S process model by defining goals as OWL-S *effects*. In addition, to support process abstraction and thus facilitate the dynamic expansion of the current workflow, the OWL-S *SimpleProcess* construct was used to define service templates within the process model that could later be expanded using the *expandsTo* object-property. This abstraction is necessary to avoid having to specify service instances at design time, but instead discover providers and their services at run-time, and thus achieving dynamic binding of services. OWL-S *preconditions* were used to represent necessary conditions for accessing a PBAC protected web service. In the event a precondition cannot be satisfied, the OWL-S service description may define an alternative set of OWL-S processes that can satisfy the original precondition. We call this alternative set a *corrective goal*.

Semantic Firewall goals can be recursive in nature. An OWL-S process that satisfies a goal (by its effect), may well have sub-goals represented by various other OWL-S workflows. The dynamic composition of these goals form a workflow, the execution of which would lead to the main goal being achieved. However, all the constituent sub-goals in the workflow would be achieved first, and may involve different actors to those directly involved in the main goal.

While exploiting these available constructs, we identified parts of the OWL-S 1.1 ontology that required extending to support PBAC policies. These extensions relate to the types of preconditions used in the SFW, which we identify as PBAC and non-PBAC specific. The main extension refers to the *PBACAccessPrecondition*, which needs to include the following properties: *testable*, *requiredRole*, *hasCorrectiveGoal* and *targetResource*. The *testable* property denotes whether a precondition can be reasonably tested before execution time; *requiredRole* specifies the PBAC role needed to successfully invoke the process; *hasCorrectiveGoal* gives an alternative goal whose effect matches the precondition; whereas the *targetResource* property refers to the semantic web service instance protected by PBAC. Because OWL-S only defines preconditions without additional parameters, we found it important to extend it to incorporate the aforementioned fields.

users	representing actors
resources	that represent service instances, each with their its own OWL-S description
local user registries	which represent the users world view

Table 1. User Components

Parameter	Instance value
testable	*true*
requiredRole	*reader-role*
correctiveGoal	*hasReadAccess*
targetResource	D_1

Table 2. PBAC Access Preconditions

Our ontology extension strategy of subclassing the *expr:SWRL-Condition* means we do not change the original semantics of the Service ontology. As can be seen, we have effectively created our own ontology with a new namespace: *pbac*. This means backwards compatibility is maintained since an OWL reasoner would understand a *pbac:SWRL-Condition* to also be an *expr:SWRL-Condition*.

4 Prototype

To evaluate the basic functionality of the Semantic Firewall, and the validity of the PBAC extensions to OWL-S, an initial prototype was built. Also, relevant insights were gained about application of the policy enforcement in multi-actor and service oriented environments.

By developing a prototype we were able to consider all the necessary components involved in the process of policy enforcement. These identified components are listed in Table 1.

Our architecture assumes that each user has its own local policy registry, which represents a user's world view. Furthermore, each user has a list of policies tuples in the form of: *(user, role, resource)* that defines what actions given user can take on a given resource. An example policy may look like: ($User_1$, *reader-role*, $dataStager_1$), which says that user with a unique identifier $User_1$ is allowed to read on a resource $dataStager_1$. As a user gains roles for different resources, their local registry is updated accordingly.

While our SFW design utilises OWL-S effect and precondition satisfaction for negotiating access to secure web services, it is important to note that this only applies to security constraints, and not to the general workflow composition problem [15]. However, the introduction of dynamic policy elements within an environment where different actors are capable of modifying existing policies according to the operations published by data service workflows requires an evaluation of the actual resource access negotiation process. For this purpose, and to evaluate a proof-of-concept study, we implemented a Java based simulation model based on the scenario described in Section 2.

The model comprises of the following components:

- A Data Service with unique identifier (D1) and two operations: `readData` and `enableRead`;
- An OWL-S description of D1 consisting of two processes: `readData` and `enableRead`, which define the operation and orchestration required to access the data from D1 depending on the actor's security role;
- Actors, which belong to a group of normal users (attempting to gain access to data from resource D1) and actors representing owners of particular resources (in this case we have the D1 Data Service owner);
- A user registry, which allows users to discover other actors (and their security roles) within the same company context and, if they have permission, update existing roles.

Given this architecture, we evaluate the access negotiation process in three different scenarios, depending on the following model setup:

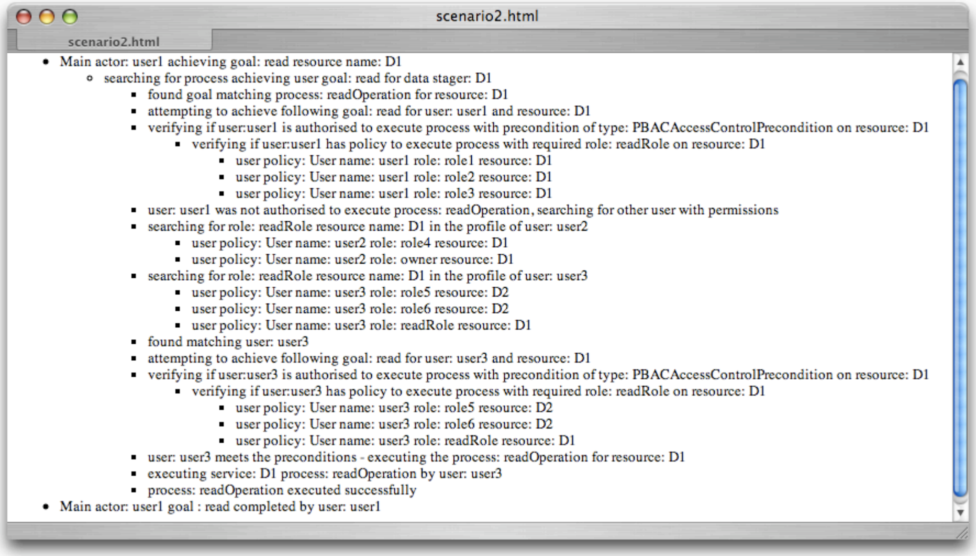
1. **Scenario 1**: where user U1 has *reader-role* on D1;
2. **Scenario 2**: where user U1 does not have *reader-role* but there exists another actor (U2) who has reader-role on D1;
3. **Scenario 3**: where user U1 does not have *reader-role* but there exists resource D1 owner who can assign *reader-role* to user U1.

In each of these scenarios it is assumed that the User (U1) attempts to perform the `read` operation on the selected data stager (D1), thus satisfying the goal `readData`. The parameters representing PBAC Access Preconditions for the `readData` process are listed in Table 2. We state that the precondition *PBACAccessPrecondition* is testable, has a property *requireRole* value *reader-role* and applies to the data stager D1. We have also specified that the property *correctiveGoal* has the value *hasReadAccess* in the event that the client discovers they cannot satisfy the *PBACAccessPrecondition*. Such a corrective goal enables the user to identify the process (i.e. the `enableRead` process in this particular case) that satisfies the *hasReadAccess* corrective goal. The matching between the corrective goal and `enableRead` process is performed by the value of the corrective goal *hasReadAccess* which is assumed to be the value of a postcondition of executing `enableRead` process for D1 resource.

Based on the Semantic Firewall algorithm (presented in Algorithm 1), the following outlines the steps encountered:

1. Given the user workflow that satisfies the goal (line 1) `readData`, the SFW inspects its processes to determine whether or not they satisfy the preconditions. Here, we are assuming that user (U_1) has found the process operating on resource D_1 that satisfies the `readData` goal.
2. If the precondition of the process is of type *PBACAccessPrecondition* and is testable (line 3), SFW will require the user's resource id, together with user policies defining what operations it is allowed to perform on which resources. Based on this, the SFW verifies whether the user (U_1) satisfies the role requirement of the given resource stated within the precondition. If user satisfies this, it is allowed to execute the process `readData`; otherwise an exception is thrown (line 5).
3. If an exception is thrown, U_1 will attempt to discover other users (relying on U_1's user registry) that are allowed to execute this process (line 8). Verification of whether a discovered user satisfies the precondition is performed by comparing the selected users policy against that required by the process precondition. If a user is found (U_2), the workflow (or process) is handed to this user, who is requested to execute it and pass the execution results to U_1 (line 10). It is up to U_2 to determine whether they will grant the request from U_1.
4. If at this point no exception has been thrown for the read process, the SFW must validate whether the effect of the process matches the initial goal. If does, then the goal has been achieved and the SFW can complete.
5. However, if an exception was thrown, indicating that other user failed to execute the workflow, U_1 extracts a workflow (line 24) from the precondition and attempts to discover (from its service registry) a process that can satisfy this goal. Because the *correctiveGoal* is an effect of the process, the search is performed by checking all the effects of the processes describing known resources until the newly satisfied effect is found. At this stage, the precondition of this process is verified, to determine whether or not the *targetResource* property refers to the resources which user is operating on. For example, the *correctiveGoal* of the `readData` process may be `enableRead`, but in order to verify whether `enableRead` would enable read on resource D_1, the *targetResource* property has to refer to D_1.

Each of the three evaluation scenarios described above were tested and found to work as expected. Figure 4 illus-


scenario2.html

scenario2.html

- Main actor: user1 achieving goal: read resource name: D1
    - searching for process achieving user goal: read for data stager: D1
        - found goal matching process: readOperation for resource: D1
        - attempting to achieve following goal: read for user: user1 and resource: D1
        - verifying if user:user1 is authorised to execute process with precondition of type: PBACAccessControlPrecondition on resource: D1
            - verifying if user:user1 has policy to execute process with required role: readRole on resource: D1
                - user policy: User name: user1 role: role1 resource: D1
                - user policy: User name: user1 role: role2 resource: D1
                - user policy: User name: user1 role: role3 resource: D1
        - user: user1 was not authorised to execute process: readOperation, searching for other user with permissions
        - searching for role: readRole resource name: D1 in the profile of user: user2
            - user policy: User name: user2 role: role4 resource: D1
            - user policy: User name: user2 role: owner resource: D1
        - searching for role: readRole resource name: D1 in the profile of user: user3
            - user policy: User name: user3 role: role5 resource: D2
            - user policy: User name: user3 role: role6 resource: D2
            - user policy: User name: user3 role: readRole resource: D1
        - found matching user: user3
        - attempting to achieve following goal: read for user: user3 and resource: D1
        - verifying if user:user3 is authorised to execute process with precondition of type: PBACAccessControlPrecondition on resource: D1
            - verifying if user:user3 has policy to execute process with required role: readRole on resource: D1
                - user policy: User name: user3 role: role5 resource: D2
                - user policy: User name: user3 role: role6 resource: D2
                - user policy: User name: user3 role: readRole resource: D1
        - user: user3 meets the preconditions - executing the process: readOperation for resource: D1
        - executing service: D1 process: readOperation by user: user3
        - process: readOperation executed successfully
- Main actor: user1 goal : read completed by user: user1

Figure 4. Output Generated when evaluating Scenario 2

trates the output generated by Scenario 2. Although this example demonstrates a simple case, it illustrates the synergy of considering an infrastructure of resources (actors, registries, etc) spanning multiple organisations (thus necessitating the use of semantic web service descriptions) that need to cooperate to enable access negotiation.

5 Discussion

Our decision to use a semantic representation for the SFW presented a challenge given that there are only a few well recognised Semantic Web Service frameworks. The main advantage of OWL-S is that it does not have a complete execution environment like that found in WSMO [14]. This means that the SFW is not constrained by the execution environment and we are able to use the OWL-S in a more flexible manner. For example, our use of OWL-S *preconditions* and *effects* goes well beyond the original intensions of OWL-S as they were intended for conditional execution rather than access control. The idea that the SFW should guide the client to achieve a goal means that the failure of a precondition should be handled, which led us to the idea of corrective goals.

We found the introduction of corrective goals, and therefore flow control within a *precondition* to be beyond the current OWL-S 1.1 specification. Coupled with the fact that different actors could achieve goals, we found it necessary to extend part of OWL-S to suite our needs. OWL-S is not the only component that could solve this problem; although it describes the goals, processes and workflow, no execution environment is provided. Our execution environment is the SFW algorithm, where the OWL-S description is used in conjunction with local user registries and different actors.

It is important to note that care has been taken to ensure this mechanism does not reveal any private details about a PBAC protected web service. For example, the access control state of the service is never transmitted to the client; neither does the client know which Certificate Authorities the web service will accept. The Semantic Firewall only describes to the client what goals exist that will give them access; such access, however, may be transient and not guaranteed over successive invocations.

6 Related work

Recent work [8], has addressed the issue of annotating service descriptions with information relating to their security requirements and capabilities. This information can then be used during the matchmaking process [17], to ensure that clients and service providers meet each others' security requirements, in addition to usual core service requirements. Such a matchmaking capability is a useful means of introducing security considerations and the ability to reason about them at the semantic level. However, the work of Denker et al. [8] focuses on describing conventional security requirements. It does not deal with how more complex information relating to security policies that interacting parties should follow are made known to potential clients, so that they can better guide the discovery process. Such work needs to be taken forward to address not just the description of conventional security requirements but also the description the related security capabilities and requirements in complex scenarios with several interacting parties and possible delegations of security capabilities or rights between services.

Work on policies, based on Semantic Web languages, provides several of the required expressive constructs for defining authorisations and obligations and their delegation [18]. Such work also takes into account some of the issues relating to conflicting policies between different do-

Algorithm 1 Semantic Firewall

discover the workflow (W_1) that meets U_1's current goal `readData()`
{*User can either state the goal or select the process that achieves the goal*}
if preconditions of W_1 are met by U_1 **then**
  if try execute W_1 **then**
    thow exception failed-precondition in W_1
  end if
else
  discover user U_2 with access permissions for services in W_1
  if U_2 exists **then**
    if try send W_1 to U_2 for execution **then**
      thow exception failed-precondition in W_1
    end if
  end if
  thow exception failed-precondition in W_1
end if

{*If no exception has been thrown, then W_1 was directly (or indirectly) executed*}
if effect **satisfies** main-goal **then**
  return
end if

while catch exception do
  find unsatisfied precondition in W_1
  if exists process/workflow that satisfies *corrective-Goal* W_2 which qualifies U_1 **then**
    {*Execute process that will provide service access to U_1*}
    if enact W_2 **then**
      thow exception failed-precondition in W_2
    end if
    {*Now service access has been granted, execute W_1*}
    if enact W_2 **then**
      thow exception failed-precondition in W_1
    end if
  end if
end while

mains, and provides means for resolving them [23]. However, although this work takes into account the existence of different policy domains, the resolution of conflicts is centrally managed and relies on basic resolution rules rather than supporting negotiation over how the conflicts can be resolved. In a centrally managed scenario this does not present a problem since the interacting parties do not need to signal their agreement with how the conflicts in policy have been resolved. However, in a scenario where each party belongs to a different organisation any resolution of conflicts should meet the approval of each interacting party. Furthermore, the deployment models suggested for policy enforcement [18, 9] may not be suitable for complex, open and dynamic environments where the interaction parties need to reason about and dynamically modify policies.

Kagal et al. [13], attempt to address some of the shortcomings identified above. They follows a more decentralised and adaptive model. The dynamic modification of policies is supported using speech acts and the suggested deployment models for this work examine different scenarios, such as FIPA-compliant agent platforms [10], web pages and web services. However, they do not take into consideration dynamic adaption of policies within the context of particular interaction scenarios to deal, for example, with notification as discussed in the example.

7 Conclusions

This paper gave an overview on the work done on the Semantic Firewall Project, which has produced a mechanism that describes a set of goals that a client must achieve to access a PBAC protected web service. The Semantic Firewall has taken a goal-based approach to describing access control policies as a set of workflows in an extended OWL-S service description. OWL-S *preconditions* were used to check necessary conditions for using the web service, while OWL-S *effects* were used as anchors to goals.

Part of our work led to several extensions of the OWL-S ontology to support conditional flow control in OWL-S preconditions. Our approach also considered the possibility of multiple actors in a complex workflow scenario, where a user may request another to invoke web services on their behalf based on security constraints.

The research is ongoing, and future work will address extending the OWL-S extensions further, to support other notions of service. Akkermans et al. [1] introduced such notions as service bundling and the sharability and consumability of resources within the OBELIX project. Current semantic web service descriptions fail to make the distinction, for example, between resources that can be copied and shared by many users, and that which is indivisible (e.g. a security credential that can only be used by one user at the time). Likewise, there is no support for establishing relationships between service elements that support each other, but are not necessarily part of a service workflow (such as representing a billing service that supports another, primary service). Future investigation will consider how such factors augment the definition of Grid services and further support policy definitions. We will also focus on formalising the notion of multiple actors within the extensions of OWL-S, to better support reasoning over several candidate *correctiveGoals* that may be found within a Virtual Organisation.

[10]http://www.fipa.org/

In addition, a better understanding and analysis is required to understand the relationship of the Semantic Firewall with OWL-WS [3, 19].

8 Acknowledgment

This research is funded by the Engineering and Physical Sciences Research Council (EPSRC) Semantic Firewall project (ref. GR/S45744/01).

References

[1] H. Akkermans, Z. Baida, J. Gordijn, N. Pena, A. Altuna, and I. Laresgoiti. Value Webs: Using Ontologies to Bundle Real-World Services. *IEEE Intelligent Systems*, 19(4):57–66, 2004.

[2] A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara. DAML-S: Web Service Description for the Semantic Web. In *First International Semantic Web Conference (ISWC) Proceedings*, pages 348–363, 2002.

[3] S. Beco, B. Cantalupo, L. Giammarino, N. Matskanis, and M. Surridge. OWL-WS: A Workflow Ontology for Dynamic Grid. In *Proceedings of the First International Conference on e-Science and Grid Computing (e-Science)*, pages 148–155, 2005.

[4] G. Berti, S. Benkner, J. W. Fenner, J. Fingberg, G. Lonsdale, S. E. Middleton, and M. Surridge. Medical simulation services via the grid. In S. Nørager, J.-C. Healy, and Y. Paindaveine, editors, *Proceedings of 1st European HealthGRID Conference*, pages 248–259, Lyon, France, Jan. 16–17 2003. EU DG Information Society.

[5] J. Blythe, E. Deelman, and Y. Gil. Automatically composed workflows for grid environments. *IEEE Intelligent Systems*, 19(4):16–23, 2004.

[6] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maquire, D. Snelling, and S. Tuecke. From open grid services infrastructure to ws-resource framework: Refactoring & evolution. Global Grid Forum Draft Recommendation, May 2004.

[7] K. Czajkowski, D. F. Ferguson, F. I, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe. The WS-Resource Framework. Technical report, The Globus Alliance, 2004.

[8] G. Denker, L. Kagal, T. Finin, M. Paolucci, and K. Sycara. Security for DAML Services: Annotation and Matchmaking. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *Proceedings of the 2nd International Semantic Web Conference*, volume 2870 of *LNCS*, pages 335–350. Springer, 2003.

[9] N. Dulay, N. Damianou, E. Lupu, and M. Sloman. A policy language for the management of distributed agents. In M. J. Wooldridge, G. Weiss, and P. Ciancarini, editors, *Agent-Oriented Software Engineering II*, volume 2222, pages 84–100. Springer-Verlag, 2001.

[10] I. Foster and C. Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2003.

[11] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. Grid Services for Distributed System Integration. *IEEE Computer*, 35(6):37–46, June 2002.

[12] J. Herveg, F. Crazzolara, S. Middleton, D. Marvin, and Y. Poullet. GEMSS: Privacy and security for a Medical Grid. In *Proceedings of HealthGRID 2004*, Clermont-Ferrand, France, 2004.

[13] L. Kagal, T. Finin, and A. Joshi. A Policy Based Approach to Security for the Semantic Web. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *2nd Int. Semantic Web Conference*, volume 2870 of *LNCS*, pages 402–418. Springer, 2003.

[14] R. Lara, D. Roman, A. Polleres, and D. Fensel. A conceptual comparison of wsmo and owl-s. In *Web Services — Proceedings of the 2004 European Conference on Web Services*, pages 254–269, 2004.

[15] S. McIlraith, T. C. Son, and H. Zeng. Semantic web service. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[16] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[17] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara. Semantic Matchmaking of Web Services Capabilities. In I. Horrocks and J. A. Hendler, editors, *International Semantic Web Conference*, volume 2342 of *LNCS*, pages 333–347. Springer, 2002.

[18] N. Suri, M. Carvalho, J. Bradshaw, M. R. Breedy, T. B. Cowin, P. T. Groth, R. Saavedra, and A. Uszok. Enforcement of Communications Policies in Software Agent Systems through Mobile Code. In *4th IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 247–250. IEEE Computer Society, 2003.

[19] M. Surridge and J. Ferris. P5.2.2 Service Binding and Access Model Registration. Technical report, ECS, IT Innovation Centre, 2006.

[20] M. Surridge, S. Taylor, D. D. Roure, and E. Zaluska. Experiences with GRIA - Industrial Applications on a Web Services Grid. In *Proceedings of the First International Conference on e-Science and Grid Computing (e-Science)*, pages 98–105, 2005.

[21] S. Taylor, M. Surridge, and D. Marvin. Grid Resources for Industrial Applications. In *2004 IEEE Int. Conf. on Web Services (ICWS'2004)*, 2004.

[22] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, D. Snelling, and P. Vanderbilt. Open grid services infrastructure. Technical report, Global Grid Forum, 2003.

[23] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. J. Hayes, M. R. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott. KAoS Policy and Domain Services: Toward a Description-Logic Approach to Policy Representation, Deconfliction, and Enforcement. In *4th IEEE Int. Workshop on Policies for Distributed Systems and Networks*, pages 93–98. IEEE Computer Society, 2003.

[24] C. Wroe, C. Goble, M. Greenwood, P. Lord, S. Miles, J. Papay, T. Payne, and L. Moreau. Automating experiments using semantic data on a bioinformatics grid. *IEEE Intelligent Systems*, 19(1):48–55, 2004.