

AGAST Final Report

Norman Gray, University of Leicester
Richard Sinnott, NeSC Glasgow
Tom Doherty, NeSC Glasgow
Jeff Lusted, University of Leicester

2009 August 20

Contents

1 Introduction

- 1.1 Deliverables p3
- 1.2 JISC pro-forma final report p3

2 Case studies

- 2.1 Astronomy: Virtual Observatory p5
 - 2.1.1 Astrogrid and access to resources p5
 - 2.1.2 The concept of community and single signon p5
 - 2.1.3 What resources and where? p5
 - 2.1.4 Applications use cases p6
 - 2.1.5 Developing the use cases p7
- 2.2 Clinical science: VOTES p8
- 2.3 Neurological science: GLASS/AvertIT p9
- 2.4 Bioinformatics: BRIDGES p10
- 2.5 Engineering: nanoCMOS p11

3 AGAST architecture

4 Implementations

- 4.1 Semantic Web and semantic-based security p13
- 4.2 Qadi: a semantic PDP p14
- 4.3 Astrogrid Common Execution Architecture p15
 - 4.3.1 Categorize application p15
 - 4.3.2 Administer community p15
 - 4.3.3 Locating the PEP: pointers within the Astrogrid architecture p16
 - 4.3.4 Ontologies p16
- 4.4 VOTES implementation p17

5 Dissemination

6 Project management and project experiences

- 6.1 Institutions and personnel p20
- 6.2 Outputs p20
- 6.3 Evaluation p21
- 6.4 Implementation experiences p22

A Technologies

- A.1 RDF, OWL, and ontologies p22
- A.2 Semantic technologies p23
 - A.2.1 Reasoners p23
 - A.2.2 Libraries and tools p24
- A.3 Privilege management infrastructures (PMI) p24
- A.4 X.812 / ISO-10181-3 p24

B Glossary

References

Executive Summary

The AGAST project – Advanced Grid Authorisation using Semantic Technologies – was funded by JISC to investigate the possibilities of making access control decisions using the technologies associated with the Semantic Web (SW), in particular using an OWL-based reasoner.

The project was driven by use-cases, elaborating a number of such use-cases across a broad spectrum of grid and e-Science projects. The project then selected two for further study. The first concerned the problem of permitting access to compute resources for various classes of astronomy users, where a key feature was the inheritance of permissions and the deduction of status from X.509 certificates, and the second concerned permitting access to clinical studies information for various classes of medical trial personnel, where a key feature was the existence of two parallel role hierarchies which had to be aligned.

The project developed a semantic ‘Policy Decision Point’ (PDP) as a REST service, and adapted two relevant ‘Policy Enforcement Point’ to act as clients of the PDP, embedded in relevant technologies. The results validated the overall design and encouraged us to make further developments. The project was reported at a major HealthGrid conference, and discussed and demoed at relevant technology workshops.

1 Introduction

The AGAST project – Advanced Grid Authorisation using Semantic Technologies – was funded by JISC to investigate the possibilities of making access control decisions using the technologies associated with the Semantic Web (SW), in particular using an OWL-based reasoner. The expectation was that this would make certain types of policy easier to express, in the sense of being logically more compact, and that they would be more readily intelligible to the policy author, and more naturally compatible with the distributed information on the Semantic Web. Although the project was informed by approaches such as PERMIS, SAML¹ and XACML², it was not a primary goal to be compatible with these.

The two high-level technical deliverables of the AGAST project were (see the glossary in Sect. B for abbreviations):

- A simple ontology for articulating policies, or, if this is more appropriate, a set of good practices for expressing policies as SPARQL queries against a generic RDF version of the relevant ACI.
- A prototype PDP and PEP (or ADF and AEF, in X.812 terms) which can make yes/no decisions based on the policy and ACI expressed in this way.

¹See <http://saml.xml.org>

²See http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

This current document is the AGAST final report, and describes the case-studies which informed our work, the architecture of the prototype PDP/PEP, the client-side implementations of the architecture, which validated our work, and a brief discussion of outputs and lessons.

1.1 Deliverables

This document comprises or describes the following AGAST deliverables:

- Deliverable 2.1: potential case-studies, and background. See Sect. 2
- Deliverable 2.2: A document describing the state of the art in semantic technologies, relevant ontologies for expressing security policy, and associated tool support. See Sect. A.1 and Sect. A.2.
- D3.1 Definition of architecture and core components. See Sect. 3.
- D3.2 Initial examples of representative policies in OWL. See Sect. 4
- D3.3 initial service prototype, comprising PDP and immediately useful translators; and D3.4 initial integration of simple PEP client with service. See Sect. 4.2
- D4.1–5 Software demonstrators for each of the noted case-studies. See Sect. 4.3 and Sect. 4.4.
- D4.6 Reports on the practical experiences of SW technologies in building these demonstrators and the consequences thereof for Grid users, developers and resource managers. See Sect. 6.2.
- D5.1 Papers for international grid conferences describing how semantic technologies can benefit Grid security; contributions to discipline-specific publication routes (for example, IVOA Notes). See Sect. 5.
- D5.2 Document describing the overall lessons learned in developing and enforcing semantic security infrastructures and their linkage to Grid infrastructures. See Sect. 6.4.
- D5.4 AGAST final report. This document as a whole.

We have not produced Deliverable D5.3 – an aspiration to hold an e-Science Institute workshop covering semantic authorisation technologies. Although the project was successful, and there is clearly substantial further progress to be made here, we did not feel that we had created a sufficient community around the technology to make such a workshop reasonable.

1.2 JISC pro-forma final report

The mapping to the JISC template final report is as follows

Acknowledgements The AGAST project was funded by the E-Infrastructure programme of JISC. The project partners were the Department of Physics and Astronomy, University of Leicester, and the National e-Science Centre Glasgow, located within the University of Glasgow.

Executive Summary See above.

Background See the introduction at the beginning of this section.

Aims and Objectives See the list of deliverables in Sect. 1.1.

Methodology The project proceeded by identifying use-cases in which existing software has to make access-control decisions, selecting two of them for further study, and adapting the existing software to make use of the access-control service we developed as part of this project. The service was provided with a RESTful interface, which limits the coupling between servers and clients. We used standard ontology libraries and specification languages.

Implementation Discussed in Sect. 6.4.

Outputs and Results The AGAST architecture is described in Sect. 3, and the server-side implementation of this, and the client-side implementations of the case-studies, are in Sect. 4.

Outcomes See Sect. 6.2 and Sect. 6.4.

Conclusions We have investigated the pragmatics of making authorisation decisions using an authorisation policy expressed as an ontology, with the expectation that this innovative approach will be a good match to the variegated landscape of ACI to be found in realistic situations on the web. We have examined a broad range of realistic use-cases, expressed the authorisation requirements of two of them using this technique, and adapted existing client applications to use the access-control framework we have developed.

We have shown that this is an adequately expressive framework for the cases we have examined, and that it is easy to use for client application authors; we additionally believe we have partially corroborated our claim that this is a usable approach for resource owners. See Sect. 6.2 for further discussion.

Implications We believe that this is a generalisable architecture, and can be further used at low cost. Since it is very explicitly intended to work in the heterogeneous environment of the web, where relevant authentication and authorisation information may be in a wide range of formats – and not necessarily expressed as authorisation information *per se* – we believe this approach is an excellent match to the problems of distributed authorisation decision-making.

Given our current results, and bearing in mind that this was a short and speculative project, we believe we have a case for a more detailed examination of more challengingly cross-system authorisation cases.

Recommendations (optional) We believe that further funding on this new authorisation approach would be necessary and useful for the general community.

References See the references section at the end.

2 Case studies

Any novel security approach based upon semantic technologies needs to be tested in realistic domains where realistically challenging policies are required, and needs to demonstrate added value, for example in supporting scenarios that cannot easily be supported by existing authorisation infrastructures. As part of the AGAST project, we identified such use-cases across a diverse range of representative e-Research applications, and in Sect. 4.3 and Sect. 4.4 describe how we used the AGAST software to implement two of these case studies.

The scenarios have been chosen to extend the existing security authorization solutions that are already in place in these projects and bring out the potential advantages that can be achieved through semantic web based approaches.

We have deliberately chosen a spectrum of research domains to cover as wide an array of scenarios as possible. These include astronomy, neuroscience, bioinformatics, clinical trials and electronic engineering. In the projects and scenarios identified below we also outline initial design plans and goals for the realization of the scenarios.

We emphasise that we did not implement all of these use-cases. We include them here, however, to act as illustrations of the sort of sophisticated and distributed access-control policies which realistic access-control methodologies must confront.

2.1 Astronomy: Virtual Observatory

2.1.1 Astrogrid and access to resources

The virtual observatory enables access to astronomical resources. Those resources are distributed across the web and are heterogeneous in nature: spectra, images, databases and applications. The resources are mediated by a service level architecture and, although access to a resource is through a single service, the architecture itself can be many-layered.

For Astrogrid the user interface to these services is via the VODesktop, or a via scripting language where defined API's exist into a service (eg: using a python API). Many things a user can accomplish can currently be undertaken *anonymously* (ie: there is no need to "sign on"). However, there are already some restrictions imposed. Access to the virtual file system (VOspace) and to some proprietary data (UKIDSS is an example) are limited: a user must "sign on" before access is enabled.

2.1.2 The concept of community and single signon

What does a user sign on to? The central concept here is one of community, which can roughly be equated to an academic department. It is at least a convenient grouping of individuals, and may be no more than that. Although it is tempting to think of a community as owning resources, it is more accurate to state that a community owns access to resources. There is a community service to support the concept and provide a single signon capability.

Astrogrid's signon capability follows an IVOA standard [8]. With the increasing development of more global VO services, it is expected the question of restricted access to resources will become a more pressing issue. The problems of authentication have largely been overcome, but there is still a need to address complex issues of authorization.

2.1.3 What resources and where?

With VO resources there are three broad areas to consider:

1. The underlying distributed filing system known as VOspace.
2. Archives of data: relational tables, images, spectra and other stores of information, which form the main stock of astronomical resources.
3. Server-based applications that process astronomical data. (Note: desktop applications will not be considered a shared resource in the current context.)

VOSpace is an underlying backbone service which can be accessed directly by an end user tool (VODesktop via a scripting API) or layered beneath one or more services (via an archive or application using a web service API). It is a complex area where scalability is a difficult issue, but nevertheless one which is still mediated by the concept of community: a person's data is held within his/her home space within their community; ie: they need to belong to a community, and therefore partake of a single signon.

Archives of data are usually accessed by some protocol with an underlying service. Access to archival data is usually open. If it is restricted it is done so on a proprietary basis.

Applications are not currently restricted in access, can consume large amounts of resources, and are mediated by the concept of community. The applications area has pioneered standards for supplying web services (see the emerging UWS standard [7]), and could in future underlie protocol implementations, allowing standardized access to archive data. Thus the existing applications framework is a fruitful area to investigate access to resources, and is taken here as a focus for analyzing some important use cases. Apart from the intrinsic value of looking at the applications area, it is likely that some valuable insights will be obtained for applying the community concept to authorization within the VOSpace and archive areas.

A possible fourth area is access to the VO registry, which is a distributed database of meta-data for VO resources. In some ways it could be considered a global resource, as registries can publish and harvest other registries across the world. This area has been ignored in the current context. Generally speaking access to registry meta-data is unrestricted.

2.1.4 Applications use cases

A user can choose an application by browsing the registry meta-data. How and why the user chooses one application over another is beyond the scope of the current exercise. The application will be located (ie: it executes) on some application server within the VO. It is not necessarily located within the local environment of the end user.

An application requires inputs and outputs. Depending upon the application, the inputs may be data provided in-stream during the invocation, or as files existing locally or within VOSpace, or all of these. The outputs may be in-stream, or provided as files existing within the application server environment or within VOSpace, or again all of these. There may well be a degree of indirection involved: the output may be an in-stream file part of whose contents points to the location of results data (within VOSpace or in the executing server environment). For a partial diagram of use cases within the applications area, see Fig. 1.

All of the bubbles within Fig. 1 are use cases; the blue coloured bubbles will be the ones focused upon in the current exercise. There are three actors involved: end user, systems administrator and job subsystem, with a central role being played by the job subsystem (a system actor). There are time considerations. The sysadm, end user and job subsystem act at different times, with only the end user and the job subsystem acting in a synchronized fashion.

The list of use cases is not exhaustive. We are limiting use cases to those associated with submitting a job for execution, with the submit job use case itself being taken as a given.

Notes:

1. There is a particular flavour to limiting access to resources within the diagram. This is the case of limiting access *before* execution begins; ie: when the job subsystem captures the job details. This is the most straightforward and perhaps the more basic and friendly approach. A variation might at-

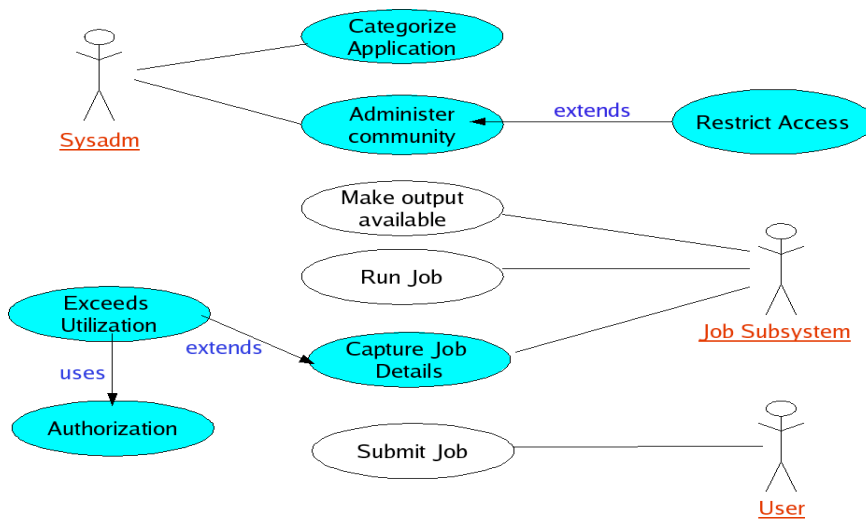


Figure 1: Application use-cases

tach some level of authorization to the execution cycle itself (for example within the run job use case: You've exceeded your CPU usage!).

2. There is a case to be made for authorizing access to job output, and for maintaining distributed job output resources. There are subtleties between access to the job log and access to a created resource. For the moment this use case will not be developed. (NB: According to Paul Harrison, this might be easy to achieve.)
3. Allied with point 2 above, the end user has been considered from the view point of the job submission use case. But it is also possible for an end user to interact with a job during the execution cycle (a: How far have we got? b: I want to cancel.). For the moment this too has not been developed.

2.1.5 Developing the use cases

The use cases in Fig. 1 naturally fall into two parts:

1. Those which are associated with the PEP in the AGAST architecture of Fig. 2. These are the use cases Exceeds Privilege and Exceeds Utilization, which are extensions of Capture Job Details, where the PEP will be located. They utilize the Authorization use case, which is equivalent to the PDP. This area will be the subject of code changes within the current Astrogrid code base.
2. Those which maintain instance data. These are the system admin tasks which allocate communities, individuals to communities, maintain relationships between groups, categorize applications according to resource usage and also for restricted or unrestricted use. Access to applications will be controlled by group (ie: community) membership. There is currently no defined utility which matches this function. The current community webapp does the critical part of maintaining communities and individuals as members, with passwords and certificates.

There is no specific use case for development of the OWL model. This will rather be the outcome of analyzing the use cases in detail.

See Sect. 4.3 for a discussion of the AGAST implementation of this use-case.

2.2 Clinical science: VOTES

The MRC funded project VOTES (*Virtual Organisations for Trials and Epidemiological Studies*, www.nesc.ac.uk/hub/projects/votes) project is building a Grid framework for clinical trials and observational studies. The project began in October 2005 and involves the National e-Science Centre at the University of Glasgow and partners at Oxford, Nottingham, Leicester and Imperial College. It is scheduled to run to March 2009. Clinical trials and clinical systems more generally place many and various demands upon security infrastructures to support the various activities of recruitment, data collection, and overall management of the trial itself.

See Sect. 4.4 for a discussion of the AGAST implementation of this use-case.

Fine grained security is essential in this context to ensure that the right data is made available to the right people for the right purpose. A key aspect of the work is that VOTES is not concerned with developing a single Grid infrastructure for a specific clinical trial or study, but with developing a Grid based framework through which a multitude of clinical trials can be supported. Versions of this framework utilizing GT4, OGSA-DAI, GridSphere and VOMS and PERMIS have already been demonstrated. In this demonstrator we will show not only how the Semantic Web can support the existing policies on access and usage of the services, but importantly, what they can add that cannot easily be supported right now.

In particular we will show scenarios such as:

- How statistical disclosure policies can be enforced through semantic web based approaches. In this scenario as part of a trial feasibility study we would like to be able to find out if there are sufficient numbers of individuals existing in clinical data resources such as the Scottish Morbidity Records and hospital data systems such as the Scottish Care Information store, i.e. patients who meet the specified criteria for potential recruitment into a given clinical trial, e.g. they should have a body mass index over a given value, had type-2 diabetes for over 2 years, been on a given drug treatment etc. Currently such queries can be run and data returned and aggregated from various clinical providers. However if there are only a few results returned from those providers then there is a danger that the identity of individuals can be revealed (statistical disclosure). This is the case for example with rare conditions. In this case, providers need to be wary of releasing their data sets, i.e. have policies on access and release. Each data provider might have their own local policies, e.g. only release pseudo-anonymised data records for patients if there are at least 10, at least 100, etc whilst the clinical trial itself might require sufficient numbers for statistical purposes (to deal with the often moderate effects of drugs etc) requires at least 1000. In this case matching of policies for data providers to deal with potential statistical disclosure policies, and clinical trials coordinators for feasibility studies needs to be reconciled. This is challenging in that the results of a given query are only known after the query is undertaken. Thus a query might be sent to one provider (NHS database) which returns 1000 results and there is therefore no need to run any further queries to other providers etc.
- Furthermore, complex diseases often have multiple contributory factors. Thus queries for diabetes may well return patients that also suffer from other diseases such as obesity or cardiovascular problems. Can we express policies that allow associations between diseases to be established? This will require to tackle the different coding systems currently in place, e.g. one site might use the International Classification of Disease version 9 whilst another uses version 10. The definition and relation between these

disease classifications through ontologies will allow for a wide variety of enhanced authorization policies to be supported.

2.3 Neurological science: GLASS/AvertIT

The JISC funded GLASS project (*Glasgow early adoption of Shibboleth*, www.nesc.ac.uk/hub/projects/glass) began in March 2006 and completed in August 2007. GLASS focused on exploring the roll-out of Shibboleth across the University of Glasgow, building upon the Novell nSure unified account management system now rolled out across campus. GLASS explored numerous scenarios where centralized authentication combined with fine grained authorization exploiting a range of virtual organization (VOrg) specific attributes were used. One of these demonstrators was within the brain trauma area working closely with the Southern General Hospital in Glasgow – specifically to support the European-wide BrainIT network (www.brainit.org). Since then these proof of concept systems have been used as the basis for the major new EU FW7 project Advanced Arterial Hypotension Adverse Event prediction through a Novel Bayesian Neural Network (AvertIT, www.avert-it.org) project. The AvertIT project involves a collaboration of 7 major neurological centres across Europe where brain trauma patients are cared for. Its goals are to better understand and ideally predict the onset of hypotensive events – a serious and life threatening problem for comatose, head injury patients.

The GLASS BrainIT demonstrator focused upon attributes delivered from a single identity provider (at Glasgow) to restrict access to federated neurological data sets. This case study will show how the semantic representation of existing security policies can be enhanced through further federated security information to decide on local access policies. Examples of the kind of policies and federated decisions we expect to explore through semantic web based technologies include the following scenarios:

- Scientists should not be both an investigator and an ethical review body member within a given neurological trial. This scenario will show how the various roles that an attribute authority can release are semantically correct according to local policy. Of course, attribute release policies can be used to only release those attributes that a site is willing to release, e.g. a given site might only release the investigator role and not the ethical oversight committee member role. This comes down to trust and agreements between federated authorities and attributes release and acceptance policies. We can also show in this scenario how someone only releasing the investigator role can be denied access if the individual (known through the distinguished name for example) is known to be an ethical oversight committee member. If the identity is not released but the investigator role is we can show how semantic technologies allow detecting non-completeness of policy specifications and subsequent decisions, and thereby how/why access decisions are denied, i.e. insufficient information was released for the policy decision to be made.
- In this case study we can also show how logical associations between roles can allow more intelligent authorisation decisions to be made through semantic technologies. Thus rather than insisting that every site supports a centrally-defined ‘investigator’ role, logical equivalents or sub-classes of investigator might also be acceptable. We will explore this flexibility in the AvertIT project where 7 neurological sites will have their own language specific policies which require such associations to be made, e.g. a policy for an AvertIT-nurse role at Glasgow should map to an AvertIT-Krankenschwester role in Heidelberg (and vice versa). Similarly if an AvertIT-doctor has a superior role to a nurse in Glasgow, then deriving that an AvertIT-Arzt

is superior to an AvertIT-Krankenschwester in Heidelberg should also be derivable and hence enforceable.

2.4 Bioinformatics: BRIDGES

The DTI funded BRIDGES project (*Biomedical Research Informatics Delivered by Grid Enabled Services*, www.nesc.ac.uk/hub/projects/bridges) completed at the end of 2005. BRIDGES developed both a large scale data Grid infrastructure linking many genomic data resources, and a compute Grid infrastructure allowing simple access to numerous compute resources including all nodes of the National Grid Service (www.ngs.ac.uk), ScotGrid (www.scotgrid.ac.uk) and local Condor pools in Glasgow to run Basic Local Alignment Search Tool (BLAST) applications. The project specified and enforced fine grained security policies (distinguishing unknown, known, and known and- trusted user groups) to restrict which computational resources the end users were allowed to access. These solutions did not require end users to have their own X.509 digital certificates, but instead used server certificates for job submission and management.

To access large-scale resources such as NGS and ScotGrid can be improved by finer grained policies, depending on (distributed) authorization and Grid environment information to schedule jobs and allocate resources. This demonstrator will show how these policies can be represented using semantic technologies, and how we can add value by showing how to do load balancing as a function of the resources already consumed by that user, that VOrg, and that institution.

In detail we will show how sites can define policies on the number of jobs that can be submitted. Thus a typical use of the BRIDGES BLAST service was where an end user uploads their sets of target queries (given in FASTA format) and these were then matched across the previously uploaded major nucleotide or protein sequence databases (typically the nt/nr data resources from NCBI). Based on the number of target sequences uploaded and their size, the BRIDGES algorithm established the appropriate number of jobs to submit and performed associated job scheduling and monitoring based on this. The actual algorithm used was naïve in this regard and for an authorized individual a set order of resources were selected, i.e. first submit to ScotGrid, then to the NGS, then to Condor pools etc until all jobs were submitted.

Through semantic web based authorization we will show how enhancements to this scenario are possible. In detail this case study will show the following capabilities:

- We will show how the BLAST algorithm can be extended to support a variety of attributes which impact upon the job scheduling algorithm, resource usage and maximize the use of the Grid more generally. Thus, a given researcher may request to submit a large batch of BLAST jobs across the Grid. Based on the local authorization policies of the distributed resources, the algorithm may need to adapt in how job submission and job scheduling is enforced. Thus the naïve approach of mapping the number of jobs onto the number of input sequences and then using this for job scheduling could be refined. A given site such as ScotGrid may have limits of jobs that can be submitted at any one time (this actually is the case and the maximum number of jobs that can be submitted at any one time is limited to 1000). Other resources such as the NGS may in turn have limits on the number of CPU hours that a user has (this is also the case where for example my allocation has been capped at 5000CPU hours), other resources such as the NeSC Condor pool has no such restrictions but offers by far the least computational resources (20PCs). These different specifications of the resource policies should inform and guide the BLAST service and its associated algorithm when dealing with different users with different privileges and the

general state of the infrastructure itself.

- Other more enhanced scenarios may also be considered where for example contention for resources exists and users with different levels of privileges exist. Knowledge of the job characteristics, the policies on access to and usage of distributed resources, and the privileges of the users themselves can be used to explore new richer algorithm extensions. Thus should the algorithm adapt to ensure that the higher privileged user gets to submit their jobs before all of the lower privileged jobs have been submitted? If there is competition for a given resource, e.g. ScotGrid then one would expect the higher privileged user to have precedence over job submission and subsequent queuing of jobs. Having different resource policies for different privileged users can raise scheduling and throughput algorithms, e.g. to allow a given user to get priority for a particular resource if it means that their combined job completes before other more privileged users who want to run larger jobs. (BLAST requires all jobs to complete before the final results can be returned – the ranking of the most similar sequences to the target genome).

2.5 Engineering: nanoCMOS

The EPSRC pilot project nanoCMOS (*Meeting the Design Challenges of nanoCMOS Electronics*, www.nanocmos.ac.uk) began in October 2006. The project is led by the University of Glasgow with the e-Science component led by NeSC Glasgow. It provides a rich vein of scenarios which will stress test the semantic web-based approaches in this proposal.

The electronics domain demands infrastructures that support IP protection, be it for designs of transistors, data sets, or quantum-mechanical simulation codes. The nanoCMOS project has developed an OMII-UK-based Grid infrastructure – providing secure access to large scale HPC resources such as the NGS – through which device level designs and simulations can be linked through to higher level circuit and system simulations, to predict the overall behaviour of systems. This scenario presents challenging and legally-charged policy decision problems concerning the access to and usage of HPC compute and data services. Specific scenarios that we intend to explore in this proposal include the following:

- It is often the case that data and simulations in this domain are not allowed to run on the NGS or other publicly shared resources for IP reasons. This is currently achieved through statically defined policies exploiting for example VOMS and PERMIS where OMII-UK GridSAM instances for job submission have policies and enforced defining which resources jobs can be submitted to (through authorization decisions deciding which DRM-Connector to use). Ideally we should be able to support dynamic changes in such policies, allowing to use new resources, remove old resources, or deal with simulations and data produced ‘on the fly’. Once such resources become available policies should be able to be dynamically updated and validated across multiple resources.
- The data associated with nanoCMOS simulations is currently being made available through the Andrew File System which itself uses KerberosV tickets. In this model, directories can be protected and shared across groups. We would like policies that allow the dynamic availability of privileges of group accounts to be captured and enforced. This can be to read, write or lock files across the various groups involved in nanoCMOS. The only way that this can be done through an authorization system right now is through definition of new roles and new authorization policies. A heavy-weight and not scalable solution. We hope to show the semantic web

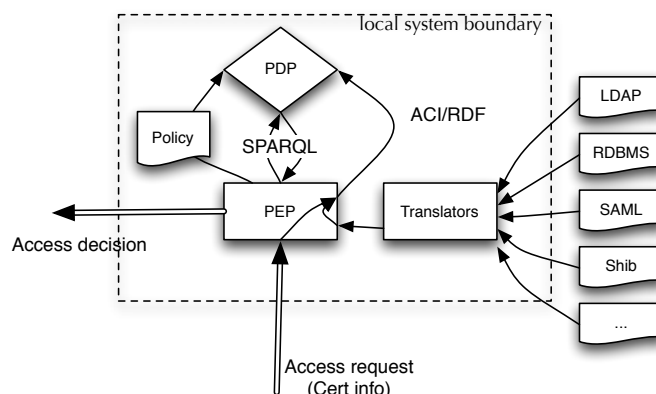


Figure 2: PDP architecture

based approaches allow a more flexible mechanism for policy definition and enforcement.

3 AGAST architecture

The AGAST architecture appears in Fig. 2

In this diagram, the PDP is a service which allows clients to use a RESTful interface to upload RDF, both (ABox) instance data and (TBox) OWL models, after which it will respond to SPARQL [6] queries against the merged model.

For example, a client might upload an assertion that an individual is at a particular university, along with information that that university is in the UK, and that the UK is in Europe; after that, it might ask (with a SPARQL ASK query) whether that individual is at a European institution. This example is worked out in detail at nwg.me.uk/note/2006/access-control/. As this example illustrates, the PDP will always answer basically the same question: is the individual of interest provably a member of the class of individuals allowed access to this resource?

We emphasise that the PDP software is entirely generic – there is nothing about the service we envisage which is specific to the problem of policy management. We believe that the PDP is largely written, in that the Quaestor server (quaestor.googlecode.com) can already do the generic work required here. Thus, the majority of the AGAST development work will probably be in the development of the PEP which uses this service, and its associated software and specifications.

The PEP is responsible for generating the ACI information which it then uploads to the PDP. It obtains this by translating information in other formats into RDF. In our overall model, we expect the PEP to be rather promiscuous in where it obtains information, taking it from LDAP servers to which it has access, or available SAML services, or even from the access request itself, if there is information within an X.509 certificate which is relevant to the policy decision: for example, an access-control policy might be such that the domain of a user's email address would be relevant ACI, and this might be reliably obtained from an access request which includes an X.509 certificate.

The access-control policy – in the sense of the precise formal specification of who is permitted access to the resource – is therefore expressed in up to three places.

1. There might be policy explicit within the ontology or ontologies used by the PEP. For example, there might be a class within the ontology named `PermittedReaders`, along with a set of necessary and sufficient conditions

under which an instance is a member of this class (in OWL terms, this is a ‘defined class’). The membership of this class might be declared to be the union of the `AtEuropeanInstitution` and `AtInstitutionFoo` classes. In this case, the access-control SPARQL query consists simply of a SPARQL ASK query, asking whether an individual `X` is a member of this `PermittedReaders` class.

2. There will be more or less policy explicit in the SPARQL query which the PEP makes of the PDP. Thus a PEP might use a SPARQL query to ask directly if `X` is a member of the `PermittedReaders` class (thus building on a policy defined in OWL), or, if such a class is not available, synthesise it by using the query to ask if `X` is in the union of the `AtEuropeanInstitution` and `AtInstitutionFoo` classes. The two cases are asking the same question, but in the first case the logic is expressed as an OWL class definition, and in the second it is expressed within the syntax of the SPARQL query.
3. In either case, the translators – which convert the heterogeneous available sources of information into RDF – must produce RDF which harmonises with the queries the PEP makes, and the OWL models which it queries against. At one extreme, a translator might assert that `X` is a `PermittedReader`, but more likely it would assert something more low-level such that `X` is a member of staff at university `Y`, doing so using the classes out of which the high-level OWL classes are composed, or the classes referred to in the PEP’s SPARQL query.

Our goal is to investigate the practicalities of this division of labour and the ideal balance between then, to discover patterns in the solutions required for our various use-cases, and to produce stereotypical SPARQL queries or OWL class definitions which will generalise our experience in this project.

Rather late in the project, we discovered the work of [4], which describes a similar approach to ours – performing access-control reasoning using OWL in the context of the Semantic Web. The approach there is more formal than ours (usefully), though we appear to have had more experience of deploying our service in practice.

4 Implementations

4.1 Semantic Web and semantic-based security

The semantic web is an intended successor to the current web. It aims to improve on the current ‘web of strings’ by specifying and deploying the technology which will let machines ‘understand’ enough that they can service our requirements better, for example by knowing that any statements made about a ‘healthcare worker’ should be immediately taken to apply also to a ‘nurse’. The technology that allows us to articulate this relationship, and state formally that a ‘nurse’ *is a* ‘healthcare worker’ is an ontology, and is key to semantic web technology.

In the context of the semantic web, ontologies are most usually expressed in the Web Ontology Language [12], which is based on the Resource Description Framework [11]. RDF is a model for representing knowledge of all types, using assertions composed of the triple of a subject, predicate and object, where subjects and predicates are named with URIs, and objects can be either URIs or literals. As an example consider the two triples:

```
<urn:example#foo> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <http://xmlns.com/foaf/0.1/Person> .
<urn:example#foo> <http://xmlns.com/foaf/0.1/name> "John Smith" .
```

These two assertions use a predicate and a class from the FOAF ontology (<http://xmlns.com/foaf/0.1/>), which is a well-known lightweight ontology for representing people and their relations and attributes. It says that the thing which has name `<urn:example#foo>` is a `foaf:Person`, and that it has `foaf:name` 'John Smith'. These can be represented somewhat more readably with the Turtle notation [1]:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
<urn:example#foo> a foaf:Person;
  foaf:name "John Smith".
```

FOAF is thus a simple ontology. It formalises well-known notions such as 'Person' and 'name', and makes extra statements about those predicates. Consider the slightly more extensive set of FOAF statements (presuming the same declaration of "foaf:" as above):

```
<urn:example#foo> a foaf:Person;
  foaf:name "John Smith";
  foaf:mbox <mailto:john@example.org>.
<urn:example#bar> foaf:mbox <mailto:john@example.org>;
  foaf:homepage <http://example.org/john>.
```

The FOAF ontology declares that the domain of the `foaf:mbox` predicate is a `foaf:Person`, and this means that a semantic web application which has ingested the set of FOAF axioms can immediately deduce that the thing named `urn:example#bar` is a `foaf:Person`, although it has not been explicitly declared as such. Another of the FOAF axioms states that at most one thing can be the subject of a `foaf:mbox` predicate with a given object. It therefore follows that `urn:example#foo` and `urn:example#bar` must be alternate names for the same thing, and thus that "John Smith" has home page <http://example.org/john>. These two examples illustrate what semantic web knowledge consists of, namely the facility for a machine to combine an ontology such as the FOAF axioms and instance data such as that given above, and deduce further instance data which is not explicit.

This example illustrates one of RDF's architectural strengths, in supporting data integration. The statements about `urn:example#foo` and `urn:example#bar` may have come from completely different sources, and be translations into RDF of very different databases, such as one organisation's LDAP service and another's personnel database. The primitiveness of the RDF model means that almost anything can be turned into RDF, and once there effectively integrated.

Given this starting point, we can promptly see how we can use this framework to support much more flexible access control policy specification and subsequent enforcement. If we decide that a particular confidential resource can be seen only by members of the class `ns1:HealthcareWorker`, which consists of the union of the classes `ns1:Doctor` and `ns1:Nurse`, and further declare that the class `ns2:Krankenschwester` is a subclass of `ns1:Nurse`, then immediately anyone declared to be of type `ns2:Krankenschwester` is provably a `ns1:Nurse` and thus also a `ns1:HealthcareWorker`. Crucially, all of the components of this chain of reasoning could potentially have come from different sources.

In this scenario, the policy information is expressed in the set of available classes, the logical relationships between them, and the grant of access to members of one of the classes.

4.2 Qadi: a semantic PDP

This architecture has been made manifest in the AGAST project through a PDP service: Qadi, which is a web service providing a RESTful interface to policy-decision services. A client of the service – most typically a PEP – may define a policy as an ontology, as we have illustrated above. When a user attempts

to access the service, the PEP obtains or extracts information about that user as RDF, and uploads that to a newly-created 'decider'. It then asks the decider "is the user X in the class of entities permitted access", to which the Qadi service can reply with a simple yes or no, which the PEP can act upon by permitting or denying the user access.

The Qadi decider accepts instance data (that is, the assertions about individual resources or users) either as RDF, or via an X.509 certificate uploaded by the client. In the latter case, the service was able to break apart the certificate, extracting the RDF assertions it implies, including for example the assertions that the X.509 principal has a given email address, or that the certificate was signed by a UK CA.

4.3 Astrogrid Common Execution Architecture

The Astrogrid project (<http://www.astrogrid.org>) is the UK participant in the international Virtual Observatory (VO) project, the International Virtual Observatory Alliance (IVOA), and has led the technical development of the project in several areas. In particular, the Astrogrid Common Execution Architecture [5] describes a framework for remote job submission, which is tuned to the type of jobs typically encountered in large-scale astronomical data management. Because of the resource implications of interacting with these resources (a catalogue query might take hours or days to run), resource owners may restrict access, or restrict run-time, to different sets of users.

4.3.1 Categorize application

A simple categorization of applications by resource usage will be used:

1. High resource user
2. Medium resource user
3. Low resource user

The principle advantage is that it is simple for the system administrator to maintain on a local basis: a judgment made on the relative importance of the typical resources consumed by an application.

4.3.2 Administer community

Currently a user can belong to a community or belong to no community. As it is a minor extension to think of an application as running within a community context (ie: from a user's view point, one that is/isn't within his/her local community), we already have three natural categories of community.

1. Local community (eg: Leicester)
2. Community (ie: any community as long as the user belongs to one)
3. No community
4. Our intention is to extend this categorization to accommodate delegation of authority: Delegated community (eg: Edinburgh when Leicester is the local community).

Communities can then be assigned access to categories of applications.

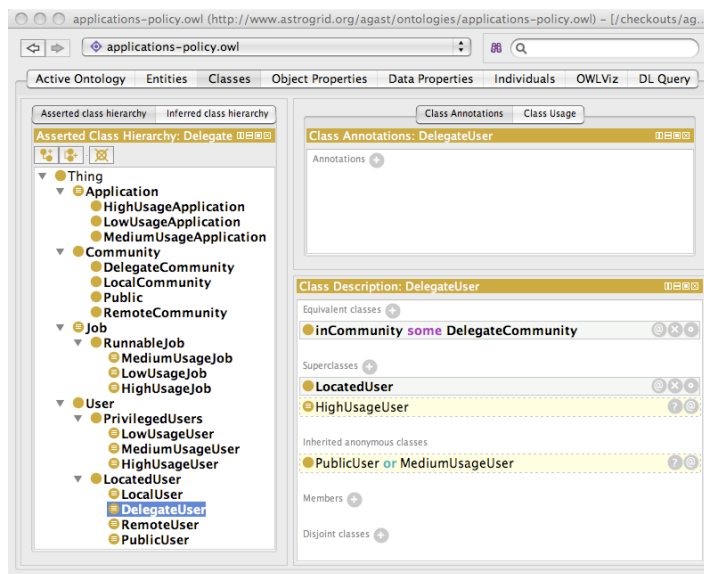


Figure 3: An ontology for quota management (Protégé screenshot)

4.3.3 Locating the PEP: pointers within the Astrogrid architecture

The CEA/CEC application server is the component dealing with managing the running of an application. The service itself can be one of two natures: a SOAP based service and a RESTful service. The latter is the basis for the UWS (Universal Worker Service) interface.

Both types of service call an implementation of a Java interface (the ExecutionController³), which contains an execute() method. The implementation of this method manages the underlying application. Intrinsic to this process is the SecurityGuard⁴. The latter has access to a users credentials and is passed an AccessPolicy⁵. An implementation of AccessPolicy will invoke the AGAST PDP. There is a good match between the AGAST architecture and the Astrogrid security architecture.

4.3.4 Ontologies

In Fig. 3 we show a representation, within Protégé, of a sample application policy, represented as an OWL ontology.

This shows a simple hierarchy of applications, communities, jobs and users. The DeLegateUser, illustrated, is defined to be one in a DeLegateCommunity (which is taken to be effectively local, and given plenty of privilege); a MediumUsageJob, for example, is defined to be one where a High- or MediumUsageUser is running a MediumUsageApplicAtion job. This ontology is part of the (local) access-control policy, as is the categorisation of applications into classes (see Fig. 4) and the translation of instance information into this RDF (potentially from other sources as illustrated in Fig. 8 but in this example entered as explicit RDF).

These definitions are fairly natural (and would be easier to write in a less generic tool), and have the effect that a MediumUsageApplicAtion being run by a LowUsageUser, for example, fails to match any of the XUsageJob definitions so that, although the result is a Job, it is not a RunnableJob. This is the case for the i:joberror-Guy-XXX and i:joberror-OJ-XX jobs in Fig. 4. The consequence

³Full package name org.astrogrid.applications.manager.ExecutionController

⁴Full package name org.astrogrid.security.SecurityGuard

⁵Full package name org.astrogrid.security.AccessPolicy

```

@prefix i: <urn:instance.data#>.
@prefix p: <urn:policy.data#>.
@prefix o: <urn:ontology#>.

# Policy information:
# declare different applications
p:appX
  a o:LowUsageApplication.
p:appXX
  a o:MediumUsageApplication.
p:appXXX
  a o:HighUsageApplication.

# ...and communities
p:uk.ac.le.star
  a o:LocalCommunity.
p:wfau.roe.ac.uk
  a o:DelegateCommunity.
p:uk.ac.cam.ast
  a o:RemoteCommunity.

# Instance information: users
i:joberror-Guy-XXX
  p:hasApplication p:appXXX;
  o:hasUser i:GuyRixon .

i:jobok-Jeff-X
  p:hasApplication p:appX;
  o:hasUser i:JeffLusted .

i:jobok-Kona-X
  p:hasApplication p:appX;
  o:hasUser i:KonaAndrews .

i:jobok-Guy-XX
  p:hasApplication p:appXX;
  o:hasUser i:GuyRixon .

i:jobok-OJ-X
  p:hasApplication p:appX;
  o:hasUser i:JohnDoe .

i:joberror-OJ-XX
  p:hasApplication p:appXX;
  o:hasUser i:JohnDoe .

# ...and their homes
i:JeffLusted
  o:inCommunity p:uk.ac.le.star .
i:KonaAndrews
  o:inCommunity p:wfau.roe.ac.uk .
i:GuyRixon
  o:inCommunity p:uk.ac.cam.ast .
i:JohnDoe
  o:inCommunity p:public .

```

Figure 4: Policy and instance data for the CEA use-case

```

prefix pol:
  <http://www.astrogrid.org/.../applications-policy.owl#>
ask {
  <urn:demo.instance.data#joberror-Guy-XXX> a pol:RunnableJob
}

```

Figure 5: A SPARQL ‘ask’ query

of *that* is that, once the ontology and instance data has been uploaded from the PEP to the Qadi PDP, the PEP needs only ask whether the job of interest is permissible, via the SPARQL [6] query of Fig. 5. This query would return a false answer from the PDP, and the job submission would therefore be refused.

4.4 VOTES implementation

As described in Sect. 2.2, the MRC-funded VOTES project (<http://www.nesc.ac.uk/hub/projects/votes>) is focused upon supporting the various phases involved in clinical trials and epidemiological studies. Fine-grained security is essential at all stages of such trial management. The VOTES project is designed to be applicable to multiple clinical trials, by developing a framework that supported the creation of multiple different clinical virtual organisations (CVOs), each with different roles/privileges that allowed access to different clinical resources. The account here is drawn from [9].

VOTES supported centralised security models and decentralised models based upon the Shibboleth technologies. The centralised model was supported through work on the VPman project (<http://www.nesc.ac.uk/hub/projects/vpman>). This project planned to show how improved Grid based security can be achieved drawing on the strengths of both VOMS and PERMIS. Specifically it wished to integrate VOMS attribute assignment function with the PERMIS authorisation de-

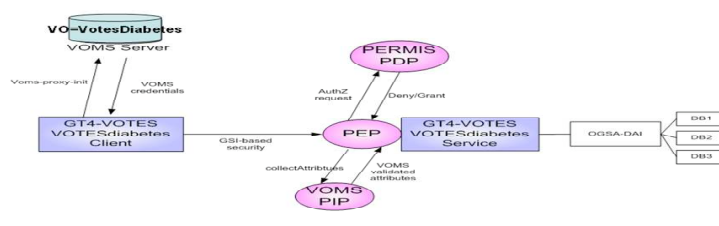


Figure 6: The GT4 VOMS-PERMIS VOTES scenario

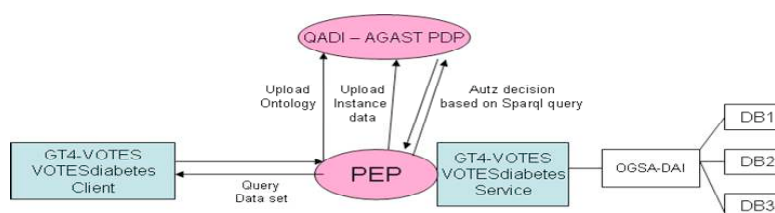


Figure 7: The AGAST VOTES scenario

cision function. This was based upon the architecture shown in Fig. 6.

In the AGAST project the architecture discussed in Fig. 2 was constructed and applied to address limitations with existing authorisation solutions as experienced in VOTES. Specifically, the AGAST-based VOTES-Diabetes VOrg replaced the PERMIS RBAC solution with the Qadi PDP, as illustrated in Fig. 7.

The interactions between these components to support semantic-based security are as follows. Firstly, a VOTES diabetes service is deployed on a GT4 infrastructure. An administrator then runs a PEP client to create a PDP instance on the Qadi service. Using the same client, the administrator then uploads an OWL ontology that acts as the security policy for the VOTES-Diabetes VOrg. This ontology consists of a hierarchy of roles and a hierarchy of access privileges associated with that trial. The administrator then uploads instance data to describe each user and their attributes. This can be in the form of an uploaded X.509 user or attribute certificate or just in RDF form. A *decider* is created. The PEP identifies the user using the DN extracted from their X.509 certificate. The PEP then interrogates the 'decider' by posting a SPARQL query and makes an authorisation decision based on the instance data that has previously been uploaded for this user. If successful the stored procedure is invoked, the federated query run and returned results joined and returned to the end user.

In both cases, users must identify themselves using X.509 certificates; there is little difference between the cases from the users' perspective. The major difference occurs with regards to policy specification and policy enforcement.

In the VOMS-PERMIS scenario, static roles *votesdiabetes-nurse* and *votesdiabetes-doctor* were created and assigned to members of the VOTES-Diabetes VOrg. This resulted in PERMIS policies being created that defined what predefined datasets each VOrg member was authorised to access. The problem is that RBAC-based hierarchies can be inflexible. Consider the real scenario where this infrastructure is used for an international Diabetes study. In this case, we may have an agreement made with a university hospital in Germany that their doctors and nurses should have the ability to query the equivalent datasets. To facilitate this we might create the roles *votesdiabetes-Krankenschwester* (Nurse) and *votesdiabetes-Arzt* (Doctor), but it is hard, in the VOMS-PERMIS scenario, to derive such horizontal equivalences, which would require a redefinition and redeployment of the static PERMIS policy.

```

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ag: <http://www.nesc.ac.uk/hub/projects/agast/rdf/x509#> .
@prefix xs: <http://www.w3.org/2001/XMLSchema#> .

[] ag:issuer [
  ag:commonName "UK e-Science CA"^^xs:string ;
  ag:countryName "UK"^^xs:string ;
  ag:distinguishedName "C=UK, O=eScienceCA, OU=Authority,
  CN=UK e-Science CA"^^xs:string ;
  ag:organizationName "eScienceCA"^^xs:string ;
  ag:organizationUnitName "Authority"^^xs:string ;
  a ag:Principal
] ;
ag:issuerAltName "support@grid-support.ac.uk"^^xs:string ;
ag:notValidAfter "2009-01-12T13:54:50Z"^^xs:string ;
ag:notValidBefore "2007-12-14T13:54:50Z"^^xs:string ;
ag:serialNumber "4008"^^xs:string ;
ag:subject [
  ag:commonName "aloyisius bloggs"^^xs:string ;
  ag:countryName "UK"^^xs:string ;
  ag:distinguishedName "C=UK, O=eScience, OU=Glasgow,
  L=Comperv, CN=aloyisius bloggs"^^xs:string ;
  ag:localityName "Comperv"^^xs:string ;
  ag:organizationName "eScience"^^xs:string ;
  ag:organizationUnitName "Glasgow"^^xs:string ;
  a ag:Principal
] ;
ag:subjectAltName "a.bloggs@physics.gla.ac.uk"^^xs:string ;
a ag:Certificate, ag:TBSertificate .

```

Figure 8: An RDF version of a proxy certificate

PERMIS is designed to integrate within the GT4 setup and has therefore quite a large installation overhead and maintenance of many configuration files, PKI for the policies and an LDAP server. Qadi has a relatively straightforward standalone setup being deployed on a Tomcat server.

This implementation exploits the Qadi translator (see Fig. 2) which ingests and translates X.509 certificates. For example, an X.509 proxy certificate is represented in RDF as illustrated in Fig. 8 (using a simple X.509 ontology Fig. 9), and the RDF information here is used within the local policy to determine access permissions.

5 Dissemination

The project website is at <http://www.nesc.gla.ac.uk/projects/agast/>.

The AGAST project and its implementations have been described in one major conference paper [9]. We have a paper submitted to All-Hands 2009.

As well, it has been discussed at several more specialised workshops, in particular the IVOA 'interoperability workshops' in September 2008 and May 2009. Although the project had anticipated (in D5.1) an intervention in the IVOA's standardisation process, it became clear that the project's outputs, though of interest to IVOA developers, were not yet mature enough for production use. The code developed for the Astrogrid CEA implementation (see Sect. 4.3) is now in the main Astrogrid code repository (though not enabled by default).

The project was discussed briefly in the D-Lib Magazine of November 2008.

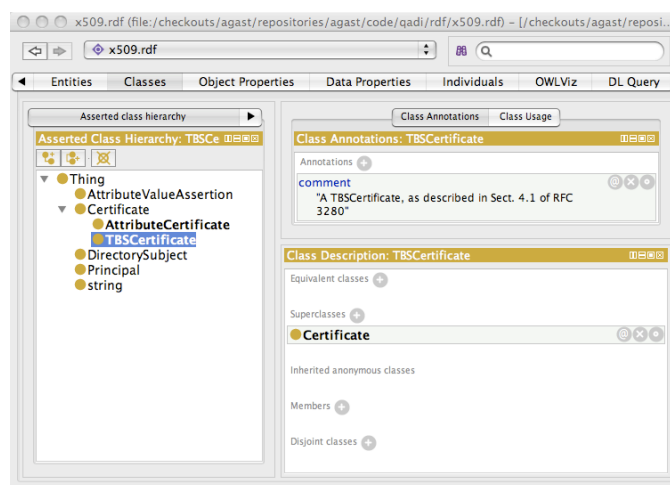


Figure 9: A simple X.509 ontology (Protégé screenshot)

6 Project management and project experiences

6.1 Institutions and personnel

The project institutions and personnel were:

University of Leicester, Department of Physics and Astronomy Norman Gray (project manager and Co-I), and Jeff Lusted

National e-Science Centre, University of Glasgow Richard Sinnott (Co-I) and Tom Doherty

The goal of the AGAST project was to produce one or more examples of a Policy Enforcement Point (PEP) which used a Policy Decision Point (PDP) implemented using semantic technologies. The project therefore began with Gray producing an initial version of such a PDP, named ‘Qadi’, based on technology originally developed for the AstroGrid project [Quaestor], and also being used in another JISC project [SKUA]. With a test bed in existence, and interfaces defined, the project then naturally fell into two distinct but interacting strands: Lusted, in Leicester, has long experience in the Astrogrid project and its software environment, and concentrated on adapting AstroGrid’s Common Execution Architecture (CEA) clients, to incorporate a PEP which used a semantic PDP; Doherty in Glasgow has wide-ranging project experience with the range of Glasgow-based use-cases, and concentrated on adapting the VOTES-Diabetes VOrg software in such a way that the policy decision was carried out by the AGAST PDP.

Because of this natural split in functionality, there was little need for an elaborate methodology, and the project was coordinated by email, following on from an initial two-day face-to-face discussion at the beginning of the project.

6.2 Outputs

For the list of deliverables, see Sect. 1.1.

The project’s stated aim was to verify that its reasoning-based PDP architecture was:

1. expressive – it is able to articulate realistic policy requirements;
2. easy to use for resource owners – it is feasible for resource owners to express their policy; and

3. easy to use for client application authors – using the Qadi service does not involve major rewrites to application software.

We believe we have achieved all of these objectives.

The first goal was achieved immediately by implementing the two use-cases in Sect. 2.1 and Sect. 2.2. These are realistic use-cases with policy requirements which have been articulated by the respective user-communities.

The third goal was demonstrably achieved by the adaptations of the respective client applications/libraries by Lusted and Doherty. The applications are existing large software sets, with pre-existing hooks for policy decision points (though not necessarily by that name). It was straightforward to adapt these to use the very simple interface which the Qadi server presented.

We believe we have at least partially corroborated the second goal. The existing tools for supporting ontology creation are very generic, and support more elaborate semantic technologies than we had to use in this project. As well, these semantic technologies are not well-known outside a particular community, and in particular the two relevant developers had had no previous experience with them. However they were able to assemble the relevant policies with little support. Having said that, for this approach to be more widespread, we would need to develop more specialised tools to support writing such policy ontologies, and probably some template ontologies (we had rather optimistically anticipated doing some work in this direction, but in the end the number of use-cases we implemented was too small for us to usefully generalise our results in this direction). We have found nothing to undermine our belief that it would be eminently reasonable to develop such tools, and that the UI metaphor which is natural to this approach – based on set inclusion rather than if-then logic – would support a tool which would be easy to use for resource administrators.

As a further outcome, we believe we have additionally demonstrated that *semantic technologies can, with well-designed services, be wrapped in such a way that significant semantic functionality can be made available to developers without them having to acquire experience in the raft of technologies that would otherwise be necessary*. As a general issue, even when a potential application is broadly discernable, there are multiple barriers of novel terminology, standards and technology to be overcome before an idea can be turned into a useful software product. This can be a significant technology hurdle for an application developer who may be rationally sceptical about the practical utility of semantic web technologies.

6.3 Evaluation

Since this was primarily a proof-of-concept project, there was no formal evaluation process. The project was successful in the most immediate sense that the Qadi-based PDP was able to provide the PDP functions required but not yet implemented in the case of CEA, and implemented using PERMIS in the case of VOTES. This was achieved with a modest software cost, and though there was insufficient time available to explore the more elaborate scenarios originally envisaged, the project's formalism was showing no signs of strain at the implementations actual performed.

A second contention in the original project proposal was that, with suitable interfaces, semantic technologies are more usable than they are sometimes painted, and indeed that there is a possible usability advantage to using the set-based semantic reasoning, over the logic-based reasoning of PERMIS. Though this could not be tested formally, without the more mature infrastructure of a follow-up to the AGAST project, it is at least corroborated by the ease with which the two project developers – both of whom had extensive expertise with the targeted case-study systems, and neither of whom had had previous experience of

semantic technologies – were able to exploit the available technology.

6.4 Implementation experiences

As discussed in Sect. 6.2, a goal of this project was to gain implementation experience with this set of technologies, and this was successful in that we believe we have verified or corroborated our main research questions.

As noted in Sect. 6.1, the software development effort naturally split into a server-side application and two adaptations of existing client-side software, produced by developers recruited for their existing expertise with the relevant applications. Since the server application was targeted at developers rather than end-users, the quality and documentation of the (RESTful) API was of primary importance, and the email and IM information flows between the various developers matched this architecture naturally and, as it turned out, effectively.

The server software, and reusable parts of the client adaptations, was hosted in a shared source code repository on a NESC machine. For minor but non-trivial technical reasons, it was infeasible to use CVS or SVN as a repository system, and after some discussion the project settled on Mercurial. This proved satisfactory, and we can warmly recommend Mercurial as a version-control (VC) system: although we took little advantage of the radically distributed possibilities which Mercurial offers, or its vaunted ease of branching, it was easy to set up a protocol which provided the advantages of more traditional VC systems, such as being able to work independently while having a secure master copy of the source; at the same time, however, it was clear that there was scope for more sophistication should we need it, without any associated fragility.

A Technologies

A.1 RDF, OWL, and ontologies

‘Semantic Technologies’ have, in one form or another, been in development for decades, arguably beginning with the interest in Artificial Intelligence in the 1950s. They have become more widely prominent in the last decade, however, with the expectation that they can be used to provide some sort of machine-readable structure layered on the Web – the Semantic Web.

The development of the Semantic Web (SW) – first described by Berners-Lee in 2001 [2] – has been assisted by standardisation efforts at the World Wide Web Consortium (W3C), most notably the definition of RDF [11], of RDFS as a simple schema language for RDF [3], and of OWL as a full-blown ontology specification language [12]. OWL is partitioned into three sub-languages of increasing expressivity: OWL-Lite is a subset which is identified as ‘easy’ to implement, OWL-DL (expressed in terms of Description Logics) is the largest subset which is computationally complete and decidable, and OWL-Full is the most expressive of the three, with semantics compatible with RDFS. The W3C OWL working group has also developed a new version of OWL, known as OWL 2. Several reasoners (Sect. A.2) in fact pragmatically define other subsets of OWL-DL.

RDF is an abstract data model, standardised by the World Wide Web Consortium [11], but drawing on a long history of Knowledge Representation work within Computing Science. There is a large collection of tutorial and overview resources at <http://www.w3.org/RDF/>. The account below is intended to fix terminology and act as a pointer to fuller descriptions; it is too brief to act as an independent tutorial.

RDF represents a broad range of assertions as sets of *triples*, consisting of *resources* named by URIs, which have *properties* whose *values* are resources or

literals. RDF, RDF Schema (RDFS) and OWL describe these resources and properties using a vocabulary which includes terms such as `rdf:type` (indicating that a particular resource is of a particular type, named by a URI), or `rdfs:subClassOf` (indicating that one named class is a sub-class, or specialisation of, another) and `owl:SymmetricProperty` (indicating that the property thus qualified is a symmetric one).

RDFS [3] adds to RDF a deliberately small set of primitive extra concepts, such as the notion of subclass and subproperty. OWL [12] adds to this a richer set of logical primitives, such as set union and intersection, and symmetric or transitive properties. The expressiveness of the RDFS and OWL languages (plus a few intermediate languages) is constrained by the requirement that the complete set of possible deductions can be derived by processors of a certain type, in non-exponential time. For example, a processor that understands RDFS will be able to deduce, from the pair of statements that `A rdfs:subClassOf B` and `B rdfs:subClassOf C`, that `A rdfs:subClassOf C`. This particular type of relationship is known, in this trade, as *subsumption*, and is key to the access-control approach described in this Note.

A collection of relations of this sort is known as an *ontology*. It consists of the definition of a number of concepts, or classes, such as `Person`, `Father` or `Sister`, together with the relationships between classes (for example `Sister rdfs:subClassOf Person`) and the definition of properties (such as `hasSibling owl:SymmetricProperty; rdfs:range Person`). When combined with assertions about individuals, such as `urn:example#Norman hasSibling urn:example#X`, the ontology allows a reasoner to derive facts about the resources which were not explicitly asserted. Given the information in this example, an OWL-aware reasoner could deduce the triples `urn:example#X a Person` (because `hasSibling` has a range of `Person`) and `urn:example#X hasSibling urn:example#Norman` (because `hasSibling` is symmetric).

A word of caution: the term ‘class’ here might be better thought of as ‘concept’. The terms ‘class’ and ‘subclass’ also appear in the context of XML Schemas, and more generally in object-oriented programming. There is an analogy between the use of the terms in those contexts and in this one, but *it is a loose analogy*: the use of types in the XSchema and O-O contexts is broadly to constrain behaviour and help identify errors, whereas the corresponding assertions in the context of RDF allow a reasoner to deduce a larger volume of implicit information. In particular, RDF schemas do not function as constraints, and mistakes made when defining concepts in an ontology, or when asserting information about resources, do not manifest themselves as ‘schema violations’, but instead more indirectly, when a reasoner finds it is able to deduce contradictory information, for example being able to prove that some resource `urn:example#X` is simultaneously a `Person` and not a `Person`.

A.2 Semantic technologies

A.2.1 Reasoners

Some ontologies exist only to provide a simple structure which an application can use to improve its interactions with users. Such simple ontologies can be used straightforwardly, as long as the application can parse the ontology format. For more sophisticated applications, however, an application needs access not only to the information explicitly asserted (`urn:example#Norman hasSibling urn:example#X` in the example above) but also to the information that that implies (`urn:example#X hasSibling urn:example#Norman`). The software which provides such services is a *reasoner*.

There are a multiple reasoners available, but three of the best-known are Pellet (pellet.owldl.org/), RacerPro (www.racer-systems.com/) and FaCT++ (code.

google.com/p/factplusplus/). These exist in library form, and packaged into services. For a fuller list, see for example en.wikipedia.org/w/index.php?title=Semantic_reasoner&oldid=300992711.

A.2.2 Libraries and tools

There are multiple libraries intended to support the use of ontologies. Of these, the best-known are Jena (jena.sourceforge.net) and Sesame (www.openrdf.org). The Qadi server is ultimately based on the Jena library.

Ontologies can be written by hand, and for simple cases this is perfectly reasonable. However for larger-scale ontologies, or ones making use of more elaborate (and syntax-heavy) OWL constructions, it is much more convenient to use a specialised tool. One of the best-known such tools is Protégé (protege.stanford.edu), which was used for the screenshots shown above.

A.3 Privilege management infrastructures (PMI)

PERMIS is an X.509 PMI. It's heavily based on LDAP servers, and turns out to be rather hard to use. The Shibboleth infrastructure involves where-are-you-from servers which exchange SAML assertions. WS-Policy, SAML, and the OASIS Extensible Access Control ML (XACML) are ways of expressing policies and assertions, as distinct from being systems which implement and reason about them. All of the systems mentioned here are, it seems, rather static, having problems with even moderately dynamic information. Related to that, though distinct, neither PERMIS nor Shibboleth does delegation very naturally (for example, 'we'll let you use this resource if institution X would let you borrow from their library').

A link between PERMIS and ontologies is described in [10], which is concerned with deducing a low-level access-control policy (for example concerning access to a single printer) from a more high-level one. There, the high-level policies are described using OWL, which is then transformed into low-level PERMIS policies using a separate rule-based system.

The work of [4] describes a system which is in some respects very similar to our own (and which we only discovered rather late in our project). Like us, they are concerned to articulate a security policy using OWL, and firmly frame the problem within the context of the Semantic Web, where the identity of the (potential) users is not necessarily known in advance.

A.4 X.812 / ISO-10181-3

ITU-T Recommendation X.812 ([13], identical to ISO/IEC-10181-3) defines a vocabulary for Access Control. This includes the usefully distinguished set of concepts below. Note that none of these are concerned directly either with authentication (the assertion that a user is who they claim to be) or with the secure transmission of assertions (such as that a given user is indeed a member of staff at a particular institution). Establishing and transmitting these assertions are problems orthogonal to authorisation, and their connection with the authorisation problem is that both are examples of ACI.

For definitions of the X.812 terms AEF, ADF, ACI and ADI, see the glossary in Sect. B.

The project use-cases are all concerned with different types of ACI, including the ACI which represents the access policies, therefore they are effectively constraints on the ADF which we must employ. Because the Access Control methodology must work with a variety of different technologies, such as web pages, JDBC calls, or RPC – that is, with a variety of different AEFs – the ADF should probably be technology neutral.

B Glossary

- AEF** *Access-control Enforcement Function*. This is the software which permits or denies access to a resource [the `/usr/bin/login` program is (as I understand it) such a function, as you have to go through this program to get access to a Unix box; it does or does not give you a shell]. That is, the AEF enforces the decision which the ADF makes [13].
- ADF** *Access-control Decision Function*. This is the function which provides the yes/no decision. The AEF consults the ADF to decide whether to grant or deny access. [In the `/usr/bin/login` case, this would I think be the `crypt(3)` function] [13].
- ACI** *Access Control Information*. This is information such as the identify of the individual requesting access (the *initiator*), their group or role memberships, or information about the properties of the *target*, such as its confidentiality. [In the `/usr/bin/login` case, this is, I think, the username/password pair] [13].
- ADI** *Access-control Decision Information*. This is the subset of the ACI which is actually used as input by the ADF. Thus the offered password plus salt is what is given to `crypt(3)` and compared with the saved password hash, so I think these are part of the ADI, but the username isn't. The time of day, or the number of previous login attempts, if they were relevant to the decision, would also be ADI, though these aren't supplied by the individual. I think that time-of-day and so on would be ACI whether or not they were used by the ADF (and thus were ADI) [13].
- IVOA** The International Virtual Observatory Alliance (www.ivoa.net) is an international consortium of national Virtual Observatory projects, which adopts a W3C-style process to broker interoperability standards between various VO projects.
- PDP** Policy Decision Point: The point where policy decisions are made. [14]
- PEP** Policy Enforcement Point: The point where the policy decisions are actually enforced [14].
- VO** Here, we take 'VO' to refer to 'Virtual Observatory', rather than 'Virtual Organisation'.
- VOrg** 'Virtual Organisation'

References

- [1] Dave Beckett. Turtle - terse RDF triple language. W3C Team Submission, January 2008. Available from: <http://www.w3.org/TeamSubmission/turtle/>.
- [2] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, May 2001.
- [3] Dan Brickley and R V Guha. RDF vocabulary description language 1.0: RDF Schema. W3C Recommendation, February 2004. Available from: <http://www.w3.org/TR/rdf-schema/>.
- [4] Lorenzo Cirio, Isabel Cruz, and Roberto Tamassia. A role and attribute based access control system using semantic web technologies. In *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops*, volume 4806 of *LNCS*, pages 1256--1266. Springer, 2007. doi:10.1007/978-3-540-76890-6_53.

- [5] Paul Harrison. A proposal for a common execution architecture. IVOA Note, May 2005. Available from:
<http://www.ivoa.net/Documents/Latest/CEA.html>.
- [6] Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF. W3C Candidate Recommendation, June 2007. Available from:
<http://www.w3.org/TR/rdf-sparql-query/>.
- [7] Guy Rixon. Introduction to CEA and UWS, version 1.00. IVOA Note, October 2007. Available from:
http://www.ivoa.net/Documents/Latest/IntroductionCEA_UWS.html.
- [8] Guy Rixon and Matthew Graham. IVOA single-sign-on profile: Authentication mechanisms, version 1.01. IVOA Recommendation, January 2008. Available from:
<http://www.ivoa.net/Documents/Latest/SSOAuthMech.html>.
- [9] Richard O Sinnott, Thomas Doherty, Norman Gray, and Jeff Lusted. Semantic security: Specification and enforcement of semantic policies for security-driven collaborations. In Tony Solomonides, Martin Hofmann-Apitius, Mathias Freudigmann, Sebastian Claudius Semler, Yannick Legré, and Mary Kratz, editors, *Healthgrid Research, Innovation and Business Case - Proceedings of HealthGrid 2009*, volume 147 of *Studies in Health Technology and Informatics*, pages 201--211. IOS Press, 2009. Available from:
<http://www.nesc.gla.ac.uk/projects/agast/HealthGrid09.pdf>,
doi : 10.3233/978-1-60750-027-8-201.
- [10] Linying Su, David W Chadwick, Andrew Basden, and James A Cunningham. Automated decomposition of access control policies. In *Sixth IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 3--13, 2005. doi : 10.1109/POLICY.2005.10.
- [11] World Wide Web Consortium. Resource Description Framework [online]. Available from: <http://www.w3.org/RDF/> [cited February 2005].
- [12] The web ontology language [online]. February 2004. Available from:
<http://www.w3.org/2004/OWL/>.
- [13] Information technology -- open systems interconnection --- security frameworks for open systems: Access control framework (ITU-T Rec X.812 = ISO/IEC-10181-3:1996). International Standard ISO-10181-3/X.812, 1996. Available from:
<http://www.itu.int/rec/T-REC-X.812-199511-I/en>.
- [14] R Yavatkar, D Pendarakis, and R Guerin. A framework for policy-based admission control. RFC2753, January 2000. Available from:
<http://www.ietf.org/rfc/rfc2753.txt>.

f612998930a1, of 2009-08-20 13:27 +0100

Copyright 2009, University of Leicester. This work is licensed under the Creative Commons Attribution-Share Alike 2.0 UK: England & Wales Licence. To view a copy of this licence, visit <http://creativecommons.org/licenses/by-sa/2.0/uk/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.