Mother, May I? OWL-based Policy Management at NASA

Michael A. Smith[1], Andrew J. Schain[2], Kendall Grant Clark[1], Arlen "Ken" Griffey[4], and Vladimir Kolovski[3]

[1] Clark & Parsia, LLC, Washington, DC
{msmith},{kendall}@clarkparsia.com
[2] NASA Headquarters, Washington, DC
andrew.schain@nasa.gov
[3] Department of Computer Science, University of Maryland, College Park, MD
kolovski@cs.umd.edu
[4] NASA Shared Services Center, Gulfport, MS
Arlen.M.Griffey@nasa.gov

Abstract. Among the challenges of managing NASA's information systems is the management (that is, creation, coordination, verification, validation, and enforcement) of many different role-based access control policies and mechanisms. This paper describes an actual data federation use case that demonstrates the inefficiencies created by this challenge and presents an approach to reducing these inefficiencies using OWL. The focus is on the representation of XACML policies in DL, but the approach generalizes to other policy languages.

1 Introduction and Motivation

NASA has at least one significant information management problem: the kind, scale, and growth-rate of its data and information is impressive and increasing. It also has significant challenges in data discovery and in the reuse and dissemination of unique scientific data collections. Several efforts are underway across NASA to use semantic technologies, including RDF and OWL, to respond to these challenges. NASA Enterprise Architects have shown considerable interest in the use of semantic technologies [1]. For additional background, see [2].

2 Requirements and Use Case

2.1 Requirements

The scale of NASA's data sources and its unique science-oriented mission require the adoption of an agile and extensible data integration platform. It is impractical for NASA to mandate that all centers and projects conform to specific institutional data models. Instead, NASA is exploring an approach that accepts data model diversity, integrating data via inter-schema mapping and alignment.

This alignment would use intermediary knowledge representations, expressed in RDF and OWL, to maximize reuse and reduce the development time of future applications. This approach and a NASA application that benefit from it are described in [2]. This paper extends that approach by recognizing that the integration challenges presented by diverse data models are accompanied by the requirement to maintain modern, manageable access control mechanisms that conform to government and industry best practices.

While data integration and maintenance are implied requirements of NASA's primary missions, they aren't the central focus.[5] This fact, and the need for fiscal responsibility, motivate the agency to use existing technology and tools for the creation, management, and enforcement of access control policies and exchange and integration of data.

Table 1 summarizes the access control policies of the constituent data sources integrated in [2]. They illustrate the nature of policies we seek to address in the short term.

Data Source	**Request**			**Decision**
	Subject	**Action**	**Resource**	
Payroll	`BrowseProcess`	Read	Any	Permit
	`ProjectManager`	Read	Any	Permit
	`Employee`	Read	$\exists about.\{\texttt{subject}\} \sqcap \neg\exists status$	Permit
	`ProjectManager`	Write	$CompetencyRecord \sqcap \exists owner.\{\texttt{subject}\}$	Permit
	`UpdateProcess`	Write	Any	Permit
	Any	Any	Any	Deny
Employee Competency	`BrowseProcess`	Read	Any	Permit
	`ProjectManager`	Read	Any	Permit
	`CivilServant`	Read	$\exists about.\{\texttt{subject}\}$	Permit
	`ProjectManager`	Write	$CompetencyRecord \sqcap \exists owner.\{\texttt{subject}\}$	Permit
	`CivilServant`	Write	$\exists about.\{\texttt{subject}\}$	Permit
	Any	Any	Any	Deny
X.500	`Employee`	Read	Any	Permit
	Any	Any	Any	Deny
Technical Reports	`Employee`	Read	Any	Permit
	Any	Any	Any	Deny

Table 1. Summary of Policies for Several NASA Data Sources

2.2 Use Case: Federated Data Access

An approach to horizontal integration of data sources that the co-authors are piloting within NASA is the development of a "global" schema that exists in a

[5] Further OMB initiatives like the Federal Transition Framework and 18 planned Federal Lines of Business further intensify NASA's motivation to quickly solve common data exchange problems within its four primary missions in order to participate in future mandated inter-agency activities.

well-defined relationship to the schema of individual data sources and the materialization of a subset of the data source data in the global schema.[6] The refresh of data in the global repository occurs at intervals determined in accordance with the requirements of the information consumer and the data source's lifecycle.[7] The federated data sources discussed here are read-only from the end user's perspective.

Integrating the data from multiple data sources presents challenges in policy management, since the data sources typically have distinct access control policies. Determining how the policies of the constituent sources align and the subsequent specification of a policy for the integrated data is a crucial aspect of this work. Formulating this alignment is a labor intensive process that is time consuming and error prone. It reduces the speed with which integrated applications can be designed and deployed and often involves personnel across organizational boundaries. The burden of the process demonstrates an undesirable tension between application deployment and security concerns. Thus, the automation of policy alignment and synthesis has the potential to increase the efficiency and security of the organization.

3 Managing XACML Policies with OWL DL

The eXtensible Access Control Markup Language (XACML)[3] is an OASIS standard language for the expression and exchange of access control policies, decision requests, and responses. In [4], co-author Kolovski introduces an algorithm for the translation of a subset of XACML into a Description Logic with the goal of offering relevant analysis services using an OWL DL reasoner, Pellet[5].

The XACML formalization includes *resources*, *subjects*, and *actions*. Considering a universe for which we seek to describe permissible activities, *subjects* are the entities which perpetrate such activities. *Actions* are a description of the nature of the activities. *Resources* are the independent entities that are the objects of such activity. Classes of *resources*, *subjects*, or *actions* can be described using general attribute value pairs, which are further discussed in section 3.2.

In its simplest form a XACML *rule* is a mapping from a set of subject, action, and resource tuples, called a *target*, to an indicator of permissibility, a *decision*, either *Permit* or *Deny*. Rules are aggregated in policies, which can be further aggregated into policy sets. At each aggregation of rules and policies, combining algorithms indicate the manner in which the collected decisions of the constituents yield a single decision for the aggregate. Combining algorithms are further discussed in the next section.

[6] The materialization of multiple databases in a global schema is frequently called data warehousing, but that term often implies a scale or business function that is irrelevant to the discussion of data federation presented here.

[7] So payroll systems are updated every two weeks, consistently; directory services are updated daily, but can be refreshed as infrequently as once per week; and technical report aggregation services are updated once or twice per year.

For reasoning purposes, [4] specifies a DL representation of XACML. Resources, subjects, and actions are represented as individuals. Targets are represented as class descriptions. For each policy and each policy set a class description exists to evaluate a *Permit* decision and a second class description exists to evaluate a *Deny* decision. Readers interested in the specifics of the XACML to DL translation are referred to [4]. In the rest of this section we introduce the elements of XACML critical for policy description with some discussion of their representation in the DL translation. Section 5 contains a summary of the policy services offered by the approach, and includes discussion of how this approach addresses the use case described above.

3.1 Combining Algorithms

Because a policy set may contain multiple policies and each policy may contain multiple rules, each of which may evaluate to different access control decisions, XACML needs some way of combining the decisions each makes. This is accomplished using a collection of combining algorithms, where each algorithm represents a different way of combining multiple access decisions into a single one. There are Policy Combining Algorithms and Rule Combining Algorithms which have similar semantics. For example, with the *Deny-overrides* algorithm, if any of the child elements return *Deny*, then the final result is also *Deny* (no matter what the other children return). The DL translation currently supports *Permit-overrides*[8], *Deny-overrides*[9], and *First-applicable*[10] combining algorithms.

3.2 Attributes and Rules

Attributes are the most basic unit of a XACML policy. They represent characteristics of the *subject*, *resource*, *action*, or *environment* in which the access request is made. It follows that access requests in XACML contain a list of attribute value pairs.

Consider an access control system integrated into the NASA Competency Management System. In this system, individual users are subjects, "read" and "write" are actions, and records describing NASA employees are resources. In such a system, a particular access request might contain a subject for which the `role` attribute has value `manager`. The basic building blocks described enable specification of a policy such as "*permit managers write access to the CMS system and deny write access to all other users.*"

Here we provide an example of a rule that returns *Deny* for access requests that have value `write` for attribute `action`. This rule would be a part of the policy described above.

[8] If any rule evaluates to Permit, then the final decision is also Permit.
[9] If any rule evaluates to Deny, then the final decision is also Deny.
[10] The effect of the first rule that applies is the decision of the policy.

```
<Rule RuleId="DefaultDenyWriteRule" Effect="Deny">
 <Target>
  <Subjects><AnySubject/></Subjects>
  <Resources><AnyResource/></Resources>
  <Actions>
   <ActionMatch MatchId="function:string-equal">
    <AttributeValue DataType="#string">write</AttributeValue>
    <ActionAttributeDesignator
      AttributeId="action"
      DataType="...#string"/>
   </ActionMatch>
  </Actions>
 </Target>
</Rule>
```

More complex policies can be created using additional XACML features. To this end, XACML reuses several XML Schema primitive datatypes and adopts some additional datatypes useful in a network access environment. Predicates for manipulation and evaluation of these datatypes adopt a similar approach—XML data standards are adopted where feasible and new predicates are defined as necessary and useful.

The DL translation provides some datatype support for values of attributes. More specifically, it offers support for built-in and user-defined XML Schema datatypes (currently only datetime and integer). For example, one could state that `age` attribute can have value ≥ 18, or that it must be one of $18, 19, 20, 21$.

3.3 Advanced XACML Features

The DL translation supports the Hierarchical Role-based Access Control Profile of XACML [6], which allows the specification of inheritance relationships between roles. For example, Role A may be defined to inherit all permissions associated with Role B. In this case, Role A is considered to be senior to Role B in the role hierarchy.

3.4 Unsupported XACML Features

The unsupported elements of XACML include multi-subject requests, complex attribute functions, rule *conditions* and some combining algorithms (see section 3.1). Some features (like complex conditions) may be impossible to analyze at development time, but there are others which we believe could be handled in the translation (some types of conditions, more expressive datatypes and the *Only-one-applicable* overriding algorithm) — all part of our ongoing work.

4 A Policy Example

This section describes a very simple example policy in more detail. Initially there are two roles, `Manager` and `Developer`; one resource: `Report`; and two

actions: `read`, `write`. The root policy set contains two policy sets which are combined using the *First-applicable* combining algorithm. The policy is presented in graphical form in figure 1.

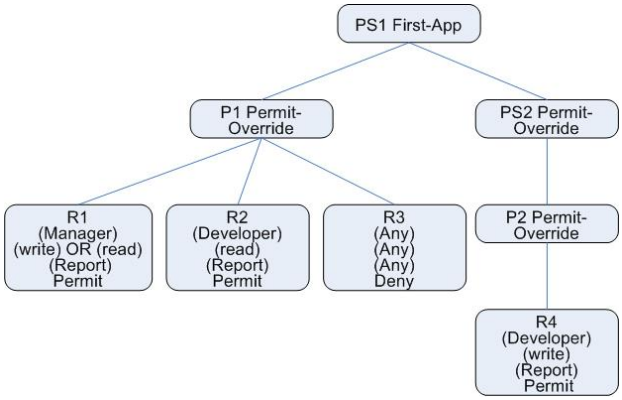

Fig. 1. Example Policy

The safety property for this example is that `Developers` cannot `write` to `Reports`. Checking the example policy against this property produces a fail, with the following counter example returned:

```
role=Manager, role=Developer, action=write, resource=report
```

Thus, if a requester is a member of both roles (`Manager` and `Developer`), then she can gain `write` access to `Report`. To prevent a `Developer` who is masquerading as a `Manager` from `writing` to `Report`, we use a *separation of duty* constraint: no user can be a member of both `Manager` and `Developer` roles at the same time.

However, the policy fails to satisfy the property even after adding the above constraint. This time, the counter example given is:

```
role=Developer, action=write, action=read, resource=report
```

Apparently there is another way for a `Developer` to gain write access: if he tries to both `read` and `write` to `Report` at the same time. To prevent this from happening, we can restrict R_2 such that *only* one value (`read`) is allowed for action attribute. After adding this constraint the policy satisfies the property.

The example policy also demonstrates redundancy. A policy element is redundant if removing the element does not change the behavior of the access policy. To motivate a redundancy detection service, consider rule R_4 in Figure 1. R_4 will always be overridden by R_3, since the policy combination is *First-applicable*, and the target of R_3 subsumes the target of R_4. In a policy evaluation engine, R_4 can

be dropped without any consequences to the security policy. This elimination of unnecessary rules at policy design- or audit-time, could provide significant optimizations for a policy framework, both in execution efficiency and human comprehensibility.

5 Policy Management Services

Some of the policy analysis services enabled by our approach are described below.

5.1 Comparison

Policy comparison is a generalized form of policy subsumption. Policy subsumption holds between two policies or policy sets, P_1, P_2, and a decision, $\alpha \in \{$ *Permit* , *Deny* $\}$, if P_1 produces decision α whenever P_2 produces decision α. Subsumption may be evaluated independently for *Permit* and *Deny*, or they may be considered together. Calculating policy subsumption is equivalent to determining DL concept subsumption.

Often knowledge of the particular requests for which the decisions differ is useful. Modifying the reasoner to generate all such instances, typically through saturation of the tableau, yields all possible requests for which the policy outcomes differ. This approach can be generalized to permit policy comparison for any decision combination.

General policy comparison is critical in a federated data environment which requires alignment of the access control policies of distinct data sources. In particular, the ability to determine the requests which yield different decisions is helpful in debugging why an individual may have a more limited view at the global data repository than at a specific constituent source. Further, using the subsumption services makes determination of the most or least restrictive policy among a set of data sources feasible, a task required when constructing a global access policy required to be as restrictive as the most restrictive of the data source policies.

Policy comparison also allows for *iterative* policy development because access control is often delegated at organizational or application boundaries. This creates an environment where local system and security administrators build policies to manage their users by refining a broader access policy. The comparison service allows these administrators to manipulate their policy and routinely evaluate its relationship to the broader policy and insure that changes don't create unintended consequences.[11]

5.2 Verification

Policy verification is the confirmation that a policy produces an expected decision for a given set of request characteristics. The mapping to DL reasoning is

[11] Difficulties associated with managing policies in a distributed system and the vulnerabilities to attack exposed by those difficulties are discussed in [7]. An alternative approach to mitigating this type of risk using statistical profiling is discussed in [8].

straightforward. Mapping the request characteristics to a class description, then determining satisfiability of the intersection of that class and the expected policy decision class description verifies the policy.

Policy verification services are essential in the NASA integrated data environment because they provide a mechanism to insure security policy compliance from design-time through audit and testing.

5.3 Creation

Representation of access control policies in OWL has additional benefits to the policy author. The general applicability of OWL permits specification of policy using terminology that exists in a broader context than the XACML document itself. A policy can adopt terminology present in enterprise data descriptions, common industry ontologies, and other OWL content. The benefit to the policy author of reusing formalisms is twofold. First, it expedites the policy creation process by avoiding repetition of already performed business process modeling work. Second, it improves quality by allowing adoption of high-quality models already validated in other deployments. Finally, use of OWL enables use of DL formalisms for natural representation of many policy idioms. Such idioms are discussed in detail in the next section. The application of OWL to policy creation and management is discussed in [9].

6 Policy Idioms

One of the distinguishing features of our approach is that the subjects, actions and resources used in the access policies are mapped to DL concepts and roles. For example, if the policy is about managers, developers, and their interaction with reports, we can have an ontology that describes the company domain, and link the policy entities with concepts in the ontology using subclass relationships. For example, we can state that a *Manager* must be an *Employee* that is the supervisor of at least one *Person*:

$$Manager \sqsubseteq Employee \sqcap \exists supervisorOf.Person$$

Using such ontologies, we show how common policy idioms can be expressed in description logics:

1. *Role hierarchies* are easily captured with subclass axioms. For example ,stating that a *LeadDeveloper* inherits all of the access privileges of the *Developer* role can be expressed as:

$$\exists role.LeadDeveloper \sqsubseteq \exists role.Developer$$

2. Hierarchies on Attributes, can be captured using property hierarchies in DL. For example, to state that if a person is a CIO of a company, that means he is also an employee of that company, we write:

$$cioOf \sqsubseteq employeeOf$$

3. *Separation of duty* constraints can be captured with disjoint axioms. To state separation of duty for two role types A and B, we use:

$$\exists role.A \sqsubseteq \neg \exists role.B$$

4. *Cardinality constraints* can be expressed on any given attribute. To state that the *role* attribute cannot have more than k values, we can write:

$$\geq k\ role.\top \sqsubseteq \bot$$

We can even specify maximum number of users that a role can have, with a combination of inverses and cardinality constraints. For example, the following says that a role cannot have more than k users:

$$\geq k\ role^{-}.\top \sqsubseteq \bot$$

7 Summary and Future Work

This paper has discussed a mechanism to represent a profile of XACML in DL with an interest in applying it to a data federation architecture at NASA. That mechanism has considerable applicability and extensibility within and outside that use case. Though not discussed here in detail, XACML has a profile targeted at representation of role based access control (RBAC)[6]. The DL formalism presented in [4] is applicable to that profile, making it a candidate for adoption in any environment that uses RBAC. At the current level of maturity, many common RBAC patterns are supported by the translation, including separation of duties and role cardinality constraints. The DL translation has potential relevance in any role based formalism and evaluation of its applicability to role-based workflow management, in particular, is an area of future research.

It is notable that although the presentation here uses XACML as the policy language, it is not a requirement of the approach. XACML was chosen because it is a standard language and has some industry adoption, but the approach detailed could be applied to any similar policy language. A notable target language is WS-Policy, a language to which DL mappings have been studied[10].

Finally, we reemphasize that the full expressivity of XACML is not yet accessible in our approach. [4] notes some constructions in XACML that are not readily translatable, though we do not know how often those constructs are used in real XACML policies. As stated in section 3, XACML uses XML Schema and additional datatypes, which yields a datatype expressivity that is beyond its counterpart in OWL DL. For this reason, user-defined datatype extensions to OWL enhance the utility of this approach by enabling translation of access control policies which specify constraints with respect to ranges of values. Such policies may reference age ranges, valid times of day, and many other common patterns. Future research is required to determine the extent of the overlap between datatype reasoning in the OWL and XACML representations.

References

1. Enterprise Architecture Data Team: Construction, collection & curation of NASA's data reference models. Technical report, NASA (Aug 2006)
2. Clark, K.G., Schain, A., Parsia, B.: Semantic web @ NASA. In: XTech 2006: "Building Web 2.0". (2006)
3. Moses, T.: eXtensible Access Control Markup Language (XACML) Version 2.0. OASIS Standard (Feb 2005)
4. Kolovski, V., Hendler, J., Parsia, B.: Analyzing web access control policies. In: 16th International World Wide Web Conference. (May 2007) To Appear.
5. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. Journal of Web Semantics (Apr 2007)
6. Anderson, A.: Core and hierarchical role based access control (RBAC) profile of XACML v2.0. OASIS Standard (Feb 2005)
7. Belokosztolszki, A., Eyers, D.: Shielding the RBAC infrastructures from cyber-terrorism. In Gudes, E., Shenoi, S., eds.: Research Directions in Data and Applications Security. Kluwer Academic Publishers (2003) 3–14
8. Bertino, E., Kamra, A., Terzi, E., Vakali, A.: Intrusion detection in RBAC-administered databases. In: ACSAC '05: Proceedings of the 21st Annual Computer Security Applications Conference, Washington, DC, USA, IEEE Computer Society (2005) 170–182
9. Rochaeli, T., Eckert, C.: RBAC policy engineering with patterns. In Kagal, L., Finin, T., Hendler, J., eds.: Proceedings of the Semantic Web and Policy Workshop. (Nov 2005)
10. Kolovski, V., Parsia, B., Katz, Y., Hendler, J.: Representing web service policies in OWL-DL. In: The Semantic Web - ISWC 2005: 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6 - 10, 2005, Proceedings (Lecture Notes in Computer Science). (2005)