Fast Semantic Attribute-Role-Based Access Control (ARBAC)

Leo Obrst[a], Dru McCandless[b], David Ferrell[a]
The MITRE Corporation
[a]McLean, VA
[b]Colorado Springs, CO
{lobrst, mccandless, ferrell}@mitre.org

***Abstract*—We report on our research effort, called Fast Semantic Attribute-Role-Based Access Control (ARBAC), to develop a semantic platform-independent framework enabling information originators and security administrators to specify access rights to information consistently and completely, in a social network environment, and then to rigorously enforce that specification. We use a modified ARBAC security model and an OWL ontology with additional rules in a logic programming and Java framework to express access policy, going beyond the limitations of previous attempts in this vein. We also experimented with knowledge compilation optimizing techniques that allow access policy constraint checking to be implemented in real-time, via a bit-vector encoding that can be used for rapid run-time reasoning.**

***Index Terms*—access control policy, attribute-based, role-based, Semantic Web, logic programming, knowledge compilation, social network, ontology, rule-based reasoning**

I. Introduction

This paper is a report of our effort to provide a semantic platform-independent framework so that information originators and security administrators can specify access rights to information consistently and completely, in a social network environment, and then to rigorously enforce that specification. In previous work [1], we discussed the architecture and some issues with optimization. In this paper, we introduce the architecture (adapted from [1]), but focus more on the optimization and implementation issues; as such, this paper can be viewed as a follow-on to [1].

For many sensitivity, privacy, and proprietary reasons, information sharing cannot be totally open. This is especially true for collaborative social environments such as the emerging MITRE Partnership Network (MPN), a large-scale environment for group-based (social network) information sharing among disparate governmental, commercial, academic, and other communities.

In addition, it is difficult to enforce unambiguous access rights and information privileges consistently and coherently and apply the access rules correctly and efficiently.

In a collaborative social environment, access control of information protecting privacy, security, and also enabling a complex range of policy respecting those requirements, is difficult.

To accomplish these objectives it is necessary to link a security policy model to a policy language with sufficient expressive power to ensure logical consistency. We used a modified Attribute-Role-Based Access Control (ARBAC) security model and an OWL ontology with additional rules in a logic programming framework to express access policy, going beyond the limitations of previous attempts in this vein, and then optimized with bit-vectors the runtime policy checking inference.

We focused on three aspects: expressivity, adaptability, and efficiency. We developed two implementations: one that transforms the policy model instance into a logic programming execution environment that includes rules; and a second that transforms the model instance into Java data structures, that in turn are optimized via a bit-encoding. In both cases, the prototype was embedded in a Java program that interfaces with external services, e.g., obtaining identity and access tokens (and their specific attribute information) from the authentication service.

The structure of the rest of the paper is as follows. In section II, we present the overall architecture and describe the runtime components. Then in section III, we briefly walk through the processing involved, followed in section IV by a discussion of the implementation. Section V addresses the optimization issues. We introduce related work in section VI, and finally, in section VII, we propose future directions.

II. System Architecture And Runtime Components

The general system architecture of the semantic ARBAC system is represented in Figure 1. It consists of three processes which flow from left to right. The three processes are: 1) the *Development* time process; 2) the *Transformation* time process; and 3) the *Execution* (runtime) process.

The Development process (the red rounded rectangle in Figure 1) involves:

1) The creation (or update) of the ARBAC ontology, represented in OWL and RDF, i.e., the semantic policy model (SPM); and

2) The instantiation of the specific ARBAC policy (policies) to be transformed and deployed, i.e., the semantic policy instance (SPI). This is an instance of the semantic policy model.

The Transformation process (the yellow rounded rectangle in Figure 1) involves developing and/or generating in Prolog and Java:

1) The transformer interpreter that will take the SPI and generate the runtime semantic policy instance (RSPI), which is the bit-vector representation of the policy + rules;
2) The attribute signature assignment engine (ASAE) which generates and updates the resource access registry (RAR);
3) The RAR, which captures the attributes of the resources in bit-vector representation, indexed by resource URI;
4) The runtime user access routine (RUAR);
5) The runtime inference engine (RTIE) which will execute the RSPI using the RUAR.

The Transformation process can thus be considered a knowledge compilation process, where source semantic models and their interpreting engines get transformed to efficient Execution time process objects.

The Execution process (the blue rounded rectangle in Figure 1) thus includes the RAR, ASAE, RTIE, and the RUAR, in addition to access to the Development and Transformation models and data.

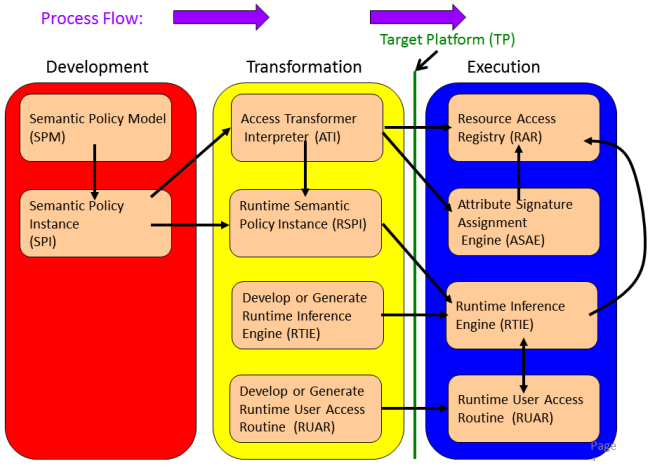

Fig. 1. Fast Semantic ARBAC System Architecture

Figure 2 displays the runtime system components of the Fast Semantic ARBAC system. The runtime system components view represents most components of the system architecture modules displayed in Figure 1, but focuses on their relationships at runtime only.

A. Semantic Policy Model (SPM)

The SPM consists of the OWL ontology classes, object properties, and data properties. The major classes consist of: *Subject* (the person, organization, software that requests specific access to a resource), *Action* (the kind of access requested, e.g., read, write, create, delete, execute, etc.), *Resource* (the object needing to be accessed by a subject: executable, graphic, text, sound, video, hardware, etc.), *Environment* (salient aspects of the space or session's environment, e.g., risk or alert level, entry network domain), *Role* (traditional roles such as administrator, expert, end user, developer, etc., that are also related to groups), and related notions: *Authentication* (how one authenticates one's identity and so, derivatively, one's potential access rights), *Security* (can span information security notions such as protocols, standards, user- and group-level passwords, encryption methods, hashing algorithms and values, etc.), *Classification Level* (proprietary, sensitive, confidential, secret, top-secret, etc.), *Identity* (Public Key Infrastructure [PKI], digital certificates, etc.), *Time* (time-stamps, time intervals with respect to various policy notions), etc.

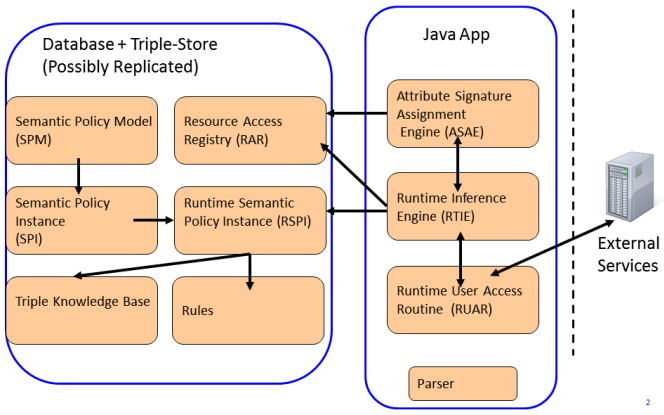

Fig. 2. ARBAC Runtime System Components

In addition, rules are a very important component of the semantic policy model (SPM). Rules exist outside of the OWL ontology per se, but are based on the classes and properties specified in the ontology. Rules were expressed initially in Prolog, and then in Java code for the second prototype. Rules are potentially recursive and express logical constraints among and across class and property values (instances). Some examples are given below.

The SPM represents a set of generic semantic components for ARBAC policy, and thus constitutes a family of potential specific ARBAC instantiations.

B. Other Components of the Architecture

For more detailed descriptions of other components of the architecture, including the SPI, RSPI, RAR, ASAE, RIE, RUAR, the OWL parser, and external service interface, we direct interested readers to [1].

III. Access Decision Process Flow and Walkthrough

The following depicts the access decision process flow.

- Initially, the Policy/Rules KB is read and loaded (including any general rules that apply to all circumstances) by the inference engine.
- Then a request comes in containing the Subject, Resource, Action, and Environment.
- The Subject's Group membership is looked up and formed.
- An initial Resource/Group/Access check may be performed.
- For some common accesses these may be cached, or may require no further processing if a quick decision can be made.
- Otherwise, the appropriate rule set is generated and populated with: any referenced access rule (pre-filtered to keep the KB small and fast), all facts about the Subject, Resource, Groups, and Environment, and General (generally applicable) rules.

- The rule set is passed to a runtime inference engine which evaluates the truth of the permission statement (something along the lines of allow(Subject, Access, Resource)).
- The Inference Engine passes back the permission decision.

The semantic policy model (SPM) is the holder of much of the underlying knowledge. Its contents include:

- Ontology
- Access Rules
- Group Membership Rules
- General Rules

The Access Rules ultimately determine whether an action can be performed on a resource (a 'Privilege' to denote the pairing of actions and resources); each rule has three parts:

1. The head, or consequence, which is always a privilege (e.g., hasPrivilege(subject22, read,medicalRecord66)). This leaves the body of the rule which for convenience is broken into 2 parts:
2. The Group membership required to obtain the privilege, and
3. Any additional requirements, expressed in terms of environment variables.

Example:

hasPrivilege(Subject, Action, Resource)

← agent(Subject), member(Subject, Group), environmentalConstraints(Group, Action, Resource, Environment), groupWithPrivilege(Group, Action, Resource, Environment).

Premises:

- All access decisions can be expressed as a *privilege* ← *requirements* rule.
- All role or subject attributes can be expressed as group membership.
- Group membership is both dynamic and contextual.
- Resources and their attributes are known a priori. If resources and attributes can change arbitrarily dynamically, this will decrease performance.

Knowledge of four things is used to resolve a permission question:

1. The Subject (the entity requesting the permission)
2. The Resource that the Subject is requesting permission about
3. The Action that the Subject wishes to perform
4. The Environment, which is a set of facts/assertions that the rules may take into account in order to make a permission determination.

The result will be either a yes or no answer as to whether permission is granted.

The access rules can have fairly complicated group membership conditions (e.g., a doctor who is an associate of a patient's primary care physician can have read access to that patient's medical record). Therefore, determining group membership may rely on a number of General Rules to help resolve the inferences (e.g., a doctor may be a member of a group; if another doctor is also a member of that group, then that doctor is an associate of the first doctor, etc.). By making group membership dynamic we can keep the access rules general.

IV. IMPLEMENTATION

The Fast Semantic ARBAC software prototype was designed to show how a system could quickly make access decisions based on the attribute values of the requesting agent. How the agent obtained the attribute values is outside the scope of the prototype; the ARBAC system is provided these from a separate source, projected to be a session authentication token (with a prescribed lifespan), that points to the attribute store, which has been obtained and encoded by the ARBAC system.

To achieve this, five conceptual classes were defined that constitute the "ARBAC view" of the world: Agents, Resources, Groups, ResourceCollections, and Policies. Two of these are collections, or sets: Groups (collections of Agents) and ResourceCollections (collections of Resources). They are hierarchical, e.g., one group may be a subset of another group, so any member of the subset group is automatically a member of the larger group. The other three classes are "flat" in an ontological sense, but contain many instances. Agents have (at least) a unique ID, and zero or more attribute/value pairs, which contain values that may be assigned to them by an organization or may be values contained in a security token. A Group is a set of Agents; group membership can be expressed in two ways: directly (an Agent by his/her ID value is asserted to be a member of a specific group) or indirectly (by specifying a set of attribute/value pairs an agent must possess in order to be a member of that group; any agent having all of the specified attribute/value pairs is considered a member of the group). Each group also has a unique ID. Unique IDs are considered special attributes and are assigned by the attribute signature assignment engine (ASAE), which updates the resource access registry (RAR). Agent IDs in the future will probably inherit the IDs of the identity token received from the external authentication service.

Resources and ResourceCollections are organized similarly to Agents and Groups. Resources also have a unique ID assigned by the attribute signature assignment engine (ASAE), and possess attribute/value pairs (such as ownedBy:: someOrganization, or locatedAt:: area). ResourceCollections likewise are sets of Resources, and membership can also be asserted directly or indirectly using a set of attribute/value pairs that a Resource must have.

Policies are different from the other four classes, in that they specify the "access rules" of what it takes for an Agent to perform some action on a Resource. In essence, a policy is just a 3-tuple containing a reference to a ResourceCollection ID that the policy controls, a reference to the Group ID to which an Agent must belong, and the action (from an enumerated set) which the Agent is requesting to perform.

The result is a simple but very flexible way to organize authorization decisions about accessing resources. In addition to general group membership, some special cases are also supported. For instance, a ResourceCollection can be created

to contain a single resource in order to directly control it. Similarly, a Group can be defined to consist of a single agent thus allowing individualized policies. Again, Groups and ResourceCollections may be organized in a hierarchy which simplifies policy creation and application. Some advanced access control mechanisms, such as an expiration date/time for an agent's token value, or the ability to specify negative conditions (e.g., agents which have a certain attribute/value pair(s) are NOT allowed access) are not implemented in this prototype, but are not precluded by this approach (i.e., they could be added at a later date without having to re-design the prototype system).

The ARBAC software is able to make quick authorization decisions because 1) most of the required information is known a priori and 2) the actual decision becomes a largely lookup-and-compare operation. The policies and resource attributes are known and stored in a location accessible to the ARBAC system. The Group and ResourceCollection definition rules are also known ahead of time and stored (although these may need to be recomputed from time to time). The agent's attribute/value pairs are passed to the ARBAC system (usually via a secureID token, but it can be done in other ways) once the agent logs onto the system. The Groups to which the Agent belongs can then be pre-computed right after login (before the Agent even selects a Resource, in most cases). Once the agent selects a Resource and the action he/she wants to take, a series of lookups take place. First, all of the policies related to the Groups to which the Agent belongs and allow the requested Action are obtained. Next, all of the IDs of the ResourceCollections to which the Resource belongs are obtained. Then the retrieved policies are examined to see if any of them contain a reference to any of the relevant ResourceCollections. If any one of them does, then that allows the Agent to access the requested Resource and perform the desired action. If none of the policies contains a reference to any of the possible ResourceCollections, then the action is not allowed.

The actual implementation of the system allows for several possibilities. Based on our work in FY12, the initial design represented each of the five conceptual classes as OWL classes, and each instance as an OWL individual. Attribute/value pairs were implemented as OWL datatype properties, as were the policy tuples. While some of the reasoning (such as class hierarchy subsumption) could be done in OWL, most of the actual policy/rule reasoning was done using Prolog. The ARBAC system converted the (hierarchically extended) information into Prolog assertions and then made a prolog query to see if a particular Agent/Resource/Action combination was allowable. While this proved workable, expressing all of the information in OWL (and using the Jena OWL reasoner to do some of the pre-computation) turned out to be somewhat cumbersome. Furthermore, the OWL format is not very interoperable with what are likely to be the other components of a true ARBAC system (such as other databases). Since only a small portion of the OWL semantics were needed, it was decided to generalize the expression of the ARBAC data by allowing it to be held in other formats, e.g., JSON (Java Script Object Notation).

Using JSON instead of OWL (with Jena) resulted in a performance increase. Also, because many data sources support JSON this approach will make interoperability much easier. Another implementation change was to use a direct bit vector approach in Java for policy evaluation, rather than Prolog. The idea is that by keeping everything in Java (Prolog requires a call to an external .dll or .so application) and using the inherent efficiency of bit reasoning, performance would increase further. So a parallel implementation using the standard Java BitSet class was created, whereby each attribute/value pair is assigned a bit position at runtime. Group membership and ResourceCollection membership were then pre-computed using a set of bits (i.e., a bit vector). When an agent selects a Resource, all of the Policies are retrieved based on the pre-computed ResourceCollections, and these are compared with the set of the Agent's Groups. If any Group is found in any of the policies, then the action is approved. Given the small set of data available, it was not possible to determine which approach (Prolog based or bit vector based, or both) will have the better performance at scale; this determination will need to be made during a follow-on test and integration effort.

V. Optimization: Bit-Encoding

Bit representation for ontology constructs (classes, properties, etc.), subsumption, and rule reasoning must address two related notions:

1) Efficiency of the representation in space and time. This includes efficiency of the encoding for storage purposes, but also compaction/compression techniques. It also includes the time required to perform the offline, development time encoding, as well as the time required to do the matching, subsumption computations, and automated reasoning performed at runtime.
2) Incremental encoding, i.e., making modifications dynamically during runtime to ontology constructs and rules, potentially recomputing the encodings of ontology constructs and rules, and then continuing efficient reasoning.

A. Ontology Constructs

The primary ontology constructs we use are the following:

- *Group:* A subclass of Collection. There are Classes of Groups (such as the Federally Funded Research and Development Center [FFRDC] class) and there are instances of Classes that are groups (e.g., the instances of the FFRDC class, such as MITRE, Aerospace, Los Alamos National Lab, etc.)
- *Resource:* A resource is any hardware, software, or service.
- *ResourceCollection:* A subclass of Collection. There are Classes of ResourceCollections and there instances of Classes that are resource collections.

- *User:* A user (agent) is generally a person, but could be a software agent.
- *Policy:* A policy is a set of access constraints on a Group or Resource created by a User who has the requisite permissions to create the policy.
- *Access:* The kind of access a User has to a Resource, as permitted by a Policy. Examples: Create, Read, Write, Delete, Execute, etc.

Because we are focusing primarily on "attributes" for access control, whether or not a User U belongs to a specific Group is a Boolean attribute, with value either 'true' or 'false' (of value 'true' if the User U is a member of a Group G, else of value 'false'). Similarly, whether or not a Resource R is a member of a ResourceCollection RG is a Boolean attribute. If it helps us in our processing, even a User U can be considered a singleton Group, i.e., a specific instance of a Group having just one member, U.

We assume a User U can create a Policy P (perhaps of a specific type) that grants another User U' specific Accesses A to a Resource R of ResourceCollection RC if the User is a member of some Group G and Group G 'owns' the ResourceCollection. Other policies may specify Roles, etc., which we are not yet addressing here.

The bit-representation for Group (and Resource) constructs is similar to the following, naïve representation:

Table 1. User Groups: Bit Representation

	G1	G2	G3	G4	G5	G6	G7	G8	G9
U1	1	1	0	0	0	0	0	0	0
U2	0	1	1	0	0	0	0	0	0
U3	0	0	1	1	1	0	0	0	0
U4	1	0	1	1	1	1	0	0	0

B. Subsumption

Subsumption is the relatively simple automated reasoning that can be done on hierarchies of classes, i.e., the taxonomic subclass 'backbone' of the ontology. These subclass hierarchies are important for ontologies, but also important for strongly typed programming languages, which perform subsumption reasoning as 'type inference' over the formal types of constructions in the specific program.

Ait-Kaci et al [4] proposed a number of bit-representations that could be used for very efficient subsumption reasoning, by plungeing the hierarchy of classes (or types), which typically constitutes a 'partially ordered set' (poset), into a boolean lattice, thus enabling efficient Greatest Lower Bound (GLB) and Least Upper Bound (LUB) operations, and efficient transitive closure. In an arbitrary poset, neither the GLB or the LUB is guaranteed to exist, but there are formal structural embeddings one can perform on the poset into an order-preserving structure, a semilattice, a lower semilattice in this initial case, which preserves the GLB, sometimes called a meet-semilattice, which says that for any nonempty finite subset of poset, there is a GLB. Note that the *ordering relation* on the elements of the poset (which define the poset) is typically notated as $\leq$, e.g., a $\leq$ b, where $\leq$ is reflexive, antisymmetric, and transitive.

An ontology *subclass* relation is an ordering relation on the classes, i.e., reflexive, antisymmetric, and transitive. OWL provides a *top* (greatest or most general) and *bottom* (least or most specific) class, called respectively Thing and Nothing, which makes OWL into a language able to model *bounded (semi-) lattices*. Bottom is often notated as $\perp$, with top notated as $\top$.

C. Encoding Bit Representations of Subsumption and Inheritance

We will discuss encodings proposed in the literature, beginning first with a naïve bit matrix representation. For all of these encodings, we adapt the example used by [17, p. 16-17], displayed in graph form as the ontology of classes in Figure 3 (where the isa relation is taken to be synonymous with the subclass relation). We use this example, rather than one drawn from our domain ontology, simply because our ontology does not currently have much depth and no multiple inheritance, which this example has. Note that these 'role' subclasses are not ontologically correct, but have been accommodated to a simple example.

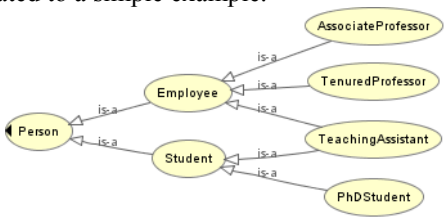

Fig. 3. Academic Role Ontology

Table 2 displays the naïve bit matrix representation for this ontology's subsumption relations. Note that the bit assignment goes as follows:

1) Initially assign 1 (true) for every class (i, j) (where i is the row, j is the column) and itself, because every class subsumes itself. This means there is a diagonal with value 1 from (1, 1) to (n, n).
2) Then for each cell of the matrix (i, j), if the class i is an ancestor of class j, assign the value 1, otherwise assign the value 0.

Table 2. Naïve bit matrix representation of Subsumption

i: row j: column	**Person**	**Student**	**Employee**	**Associate Professor**	**Tenured Professor**	**PhD Student**	**Teaching Assistant**
Person	1	1	1	1	1	1	1
Student	0	1	0	0	0	1	1
Employee	0	0	1	1	1	0	1
Associate Professor	0	0	0	1	0	0	0
Tenured Professor	0	0	0	0	1	0	0

PhD Student	0	0	0	0	0	1	0
Teaching Assistant	0	0	0	0	0	0	1
⊥	0	0	0	0	0	0	0

This encoding thus is the reflexive, transitive closure of the (antisymmetric) subclass (isa) hierarchy of Figure 4.

The naïve bit-assignment algorithm as represented in Table 2 is bottom-up, with an implicit 'bottom' (⊥). The classes Employee and Student, and then Person, are the only classes which have subclasses.

Subsumption between two classes can then be computed in constant time using a binary AND operation on the bit vectors of the two classes. The subsumption operator over the bit-encoded classes is defined as follows.

Definition: Subsumption over Bit-Encoded Classes:

Let $x_1, \ldots, x_n$ be classes in a subclass hierarchy, γ be an bit-encoding function, and $\sqsubseteq$ be the *subsume* relation (where α, β are classes and $\alpha \sqsubseteq \beta$ is read as 'class α subsumes class β'):

Then the following holds:

i. $\gamma(x_i) \sqsubseteq \gamma(x_j) \leftrightarrow \gamma(x_i)$ AND $\gamma(x_j) = \gamma(x_j)$

[the encoding of the first class subsumes the encoding of the second class if and only if the binary AND of those encodings is equal to the encoding of the second class]

ii. $\gamma(x_j) \not\sqsubseteq \gamma(x_i) \leftrightarrow \gamma(x_j)$ AND $\gamma(x_i) \neq \gamma(x_j)$

[the encoding of the first class does not subsume the encoding of the second class if and only if the binary AND of those encodings is not equal to the encoding of the second class]

Example 1: Does TeachingAssistant subsume AssociateProfessor?

I.e., does AssociateProfessor occur in the transitive closure of the subclass relation of TeachingAssistant?

SubsumeS (TeachingAssistant, AssociateProfessor)

= AND (0000001, 0001000) = 00000000, i.e., no.

Example 2: Does Person subsume TeachingAssistant?

Subsumes (Person, TeachingAssistant)

= AND (1111111, 0000001) = 0000001, i.e., yes, because the result 0000001 = 0000001 (the encoding for TeachingAssistant.

Example 3: Does Employee subsume Student?

Subsumes (Employee, Student)

= AND (0011101, 0100011) = 0000001, i.e., no, because the result 0000001 ≠ 0100011 (the encoding for Student).

What if one wants at runtime to add a new class incrementally (dynamically) after the above bit-representation has been generated at development time? We add the new class ResearchAssistant to the original ontology, resulting in Figure 4.

Recomputing our bit-matrix, we arrive at the following, Table 3. Note that we have to add a new bit by creating a new row and new column for ResearchAssistant, which we add as a new i+1 row and a new j+1 column into the matrix (but above the implicit □).

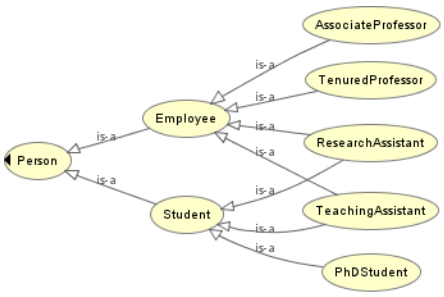

Fig. 4. Academic Role Ontology + ResearchAssistant

If we added the new bit as a new row and new column at the beginning of the matrix, then we would maintain the 1-bit diagonal we saw in Table 2. In addition, of course, we have to update the entries in the new Research Assistant column with their values (1 if an ancestor of Research Assistant, 0 otherwise). The naïve bit-encoding of Subsumption requires n2 bits.

Table 3. Naïve bit matrix representation of Subsumption with Incrementally Added ResearchAssistant Class

i: row j: column	**Research Assistant**	**Person**	**Student**	**Employee**	**Associate Professor**	**Tenured Professor**	**PhD Student**	**Teaching Assistant**
Person	1	1	1	1	1	1	1	1
Student	1	0	1	0	0	0	1	1
Employee	1	0	0	1	1	1	0	1
Associate Professor	0	0	0	0	1	0	0	0
Tenured Professor	0	0	0	0	0	1	0	0
PhD Student	0	0	0	0	0	0	1	0
Teaching Assistant	0	0	0	0	0	0	0	1
Research Assistant	1	0	0	0	0	0	0	0
⊥	0	0	0	0	0	0	0	0

Ait-Kaci et al [4] propose a number of new methods for encoding subsumption. Their first method requires a bottom-up (from the terminal classes to the root class) computing of the binary OR of the bits assigned to children classes, the result of which becomes the bit-encoding of their parent classes. New bits are introduced whenever a parent has just one class and

whenever a false positive subsumption would result. If incremental updates to the encoding are necessary, there are potential complications. If one wants to add new leaf (terminal) class nodes to the hierarchy, such as we did with ResearchAssistant above, there are no issues. However, if one wants to add new non-terminal (or root) nodes, there are complications. If a class C_j is added that has the same inheriting subclasses as an existing class C_i, then a new bit must be added to re-encode the existing class and all of its ancestors too. In addition, any new non-terminal class will have to have the ancestors of its children classes checked for conflicting encodings.

For a discussion of other bit-encoding techniques, the interested reader is directed to [17, pp. 16-23]. There are other encoding approaches, including interval-encodings. Interval-based encodings compute non-overlapping codes for the children within the interval of the parent, but do not support multiple inheritance.

In fact, although each of the above approaches out-perform the naïve encoding, all of them have some issues (except perhaps [17], which relies on binary representation of prime numbers) with incremental (dynamic) updates, requiring some recomputation of encodings and determination of conflicts, which in turn may require recomputation of encodings.

Rules too may be given encodings, but space limitations preclude a discussion of this topic here, but see [8] for Boolean satisfiability (SAT) reasoning using bit-matrices.

VI. Related Work

There is much previous related research across multiple dimensions (access control regimes, policy languages and approaches, specialized languages (and logics) vs. ontology approaches, knowledge compilation issues, bit-vector and other optimization approaches, social network approaches, privacy vs. security issues and approaches, etc.) that have influenced our current and impending work.

In order to accomplish our objectives it was necessary to link a security policy model to a policy language with sufficient expressive power to ensure logical consistency. We extend the NIST Role-Based Access Control (RBAC) security model [15] and related approaches [18-19], as have many other researchers to include attributes, and extend the Web Ontology Language (OWL) with additional rules to express access policy using logic programming, and beyond the limitations of [20]. Unfortunately, given our own space limitations here, we cannot do an extensive comparison of our approach across the multiples dimensions with other approaches, nor justly describe those other approaches.

In addition, there is extensive research in more general policy-based approaches that could be employed also for access control [21-22].

There are other Semantic Web-based approaches (including [22]), some of which address more specifically social network types of applications [23, 24].

For implementation in real-time, via a bit-vector or other efficient encodings that can be used for rapid run-time reasoning, we've looked at [2-6, 7-12, 17]. For bit-vector representation to support RDF triples, we investigated [11-14].

Our own previous work addressed issues in translating OWL/RDF ontologies and Semantic Web Rule Language Rules (SWRL) [25] into logic programming for efficient runtime reasoning, and employing knowledge compilation techniques [26-28], which we also generalized to address services using first-order logic theorem provers and for ontology alignment [29].

VII. Future Work

Although we have investigated and implemented some optimizations, e.g., extensionalization and delayed rule evaluation, we have only rudimentarily implemented the second-level of optimization we intended, i.e., the bit-representation execution at runtime.

If we had additional time, we intended to implement the prime-number bit-encoding of subsumption described in [17]. In general, for the restricted reasoning we need for access control policy enforcement as described in this paper, and given the probable volume of access request determinations (and thus subsumption and equivalence checks, rule execution) we foresee needing in a complex collaborative social network environment such as the MPN, optimized efficient automated reasoning is necessary. Traditional, more general description logic reasoners were deemed too slow (Pellet, etc.) In addition, most proposed bitmap encodings for subsumption and type reasoning are efficiently statically initialized and then used, but dynamically updating the subsumption/type hierarchy, i.e., adding, deleting, modifying classes and properties (which will happen, under the Open World Assumption of OWL and first-order logic), leads to degraded performance and increasingly baroque re-encodings to avoid conflicts.

Therefore, we would consider implementing the bit-encoding scheme based on assigning prime numbers to nodes in the class and property subsumption graphs, as developed by Preuveneers and Berbers [17, 30]. Adding a new class or property does not require re-encoding. Furthermore, the encoding automatically provides us the direction of the relationship. Modular hierarchies, each separately encoded, with very efficient subsumption-checking, are the result. Figure 5 depicts a subclass hierarchy encoded using prime numbers.

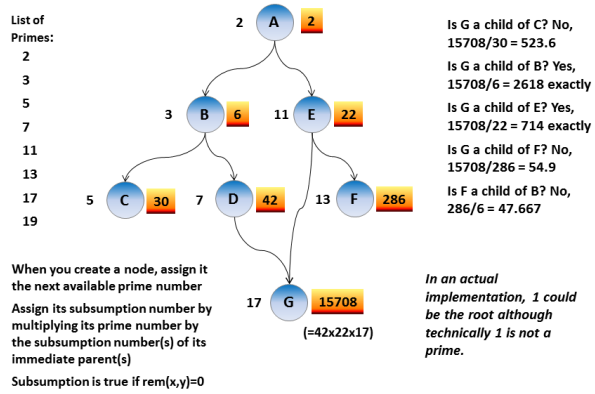

Fig. 5. Prime Number Encoding for Class Subsumption

In addition to the use of prime numbers, the scheme of [17, 30] defines a compact binary matrix representation of the inheritance relationships, which we will not go into here.

Evaluation done in [30, p. 32] shows that subsumption testing in his scheme is much faster than that of some major existing description logic reasoners, on the order of 250 times faster than Pellet. An evaluation performed on a different project we are involved in, written in C/C++ demonstrated 1000% improvement using this method of subsumption checking over the previous naïve, breadth-first search of the subsumption graph.

ACKNOWLEDGMENT

REFERENCES

[1] Obrst, L.; D. McCandless; D. Ferrell. 2012. "Fast Semantic Attribute-Role-Based Access Control (ARBAC) in a Collaborative Environment." The 7th IEEE International Workshop on Trusted Collaboration (TrustCol 2012), October 14–17, 2012, Pittsburgh, PA.

[2] Abadi, D. J.; A. Marcus; S. Madden; K. J. Hollenbach. 2007. "Scalable Semantic Web Data Management Using Vertical Partitioning." In Proceedings of VLDB, pages 411~422, September 2007.

[3] Ait-Kaci, H. 1984. "A Lattice-Theoretic Approach to Computation Based on a Calculus of Partially-Ordered Type Structures." Ph.D thesis, Computer and Information Science Dept., Univ. of Pennsylvania, Philadelphia, PA.

[4] Ait-Kaci, H.; R. Boyer; P. Lincoln; R. Nasr. 1989. "Efficient Implementation of Lattice Operations." TOPLAS 11-1-1989.

[5] Blandford, D. K.; Blelloch, G. E.; and Kash, I. A. 2003. "Compact representations of separable graphs." Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (Baltimore, Maryland, January 12 - 14, 2003).

[6] Blandford, D. K.; Blelloch, G. E.; and Kash, I. A. 2004. "An Experimental Analysis of a Compact Graph Representation." In Proceedings of ALENEX04.

[7] Caseau, Y.; M. Habib; L. Nourine; O. Raynaud. 1999. "Encoding of multiple inheritance hierarchies and partial orders." Computational Intelligence 15 (1), 50-62.

[8] Dershowitz, N. 2008. "Bit Inference." Workshop on Practical Aspects of Automated Reasoning, August, 2008, Sydney. 26-35.

[9] Fall, A. 1995. "Heterogeneous Encoding." In Proceedings of International KRUSE Symposium: Knowledge Retrieval, Use, and Storage for Efficiency, Gerard Ellis, Robert Levinson, Andrew Fall, Veronica Dahl, eds., Santa Cruz, CA, Aug. 11-13, pp. 134-146 (1995).

[10] Krall, A.; Vitek, J., Horspool, 1997. "Near optimal hierarchical encoding of types." 11th European Conference on Object Oriented Programming (ECOOP'97). Springer (1997).

[11] McGlothlin, J. P.; L. Khan, B. Thuraisingham. 2011. "RDFKB: A Semantic Web Knowledge Base." IJCAI, 2011.

[12] McGlothlin, J. P.; L. Khan. 2008. "RDFVector: A Scalable Data Model for Efficient Querying of RDF Datasets." http:// www.utdallas.edu/~jpm083000/ssDBM.pdf.

[13] McGlothlin, J.P.; L. Khan. 2010b. "Efficient RDF data management including provenance and uncertainty." IDEAS, 193-198, August 2010.

[14] McGlothlin, J. 2010. "RDFVector: An Efficient and Scalable Schema for Semantic Web Knowledge Bases." PhD Symposium, 7th Extended Semantic Web Conference (ESWC 2010), Heraklion, Greece. May 30 – June 3, 2010..

[15] http://csrc.nist.gov/groups/SNS/rbac/.

[16] Neumann, T.; G. Weikum. 2009. "RDF-3X: a RISC-style engine for RDF." In Proc. of VLDB, pages 647-659, September 2009.

[17] Preuveneers, D.; Berbers, Y., 2006. "Prime numbers considered useful: Ontology encoding for efficient subsumption testing," Tech. Rep. CW464. http://www.cs.kuleuven.be/publicaties/rapporten/cw/CW464. Department of Computer Science, Katholieke Universiteit Leuven, Belgium (October 2006).

[18] Sandhu, R. 1998. "Role-based access control." In M. Zerkowitz, editor, Advances in Computers, volume 48. Academic Press.

[19] Sandhu, R.; E. J. Coyne; H. L. Feinstein; and C. E. Youman. "Role-based access control models." 1996. IEEE Computer, 29(2):38–47, February 1996.

[20] Finin, T.; A. Joshi; L. Kagal; J. Niu; R. Sandhu, W. Winsborough; and B. Thuraisingham. 2008. "ROWLBAC: representing role based access control in OWL." In Proceedings of the 13th ACM symposium on Access control models and technologies (SACMAT '08). ACM, New York, NY, USA, 73-82.

[21] Tonti, G.; J. M. Bradshaw; R. Jeffers, R. Montanar; N. Suri; and A. Uszok. 2003. "Semantic web languages for policy representation and reasoning: A comparison of kaos, rei, and ponder." 2nd International Semantic Web Conference (ISWC2003). Springer-Verlag.

[22] Uszok, A.; J.M. Bradshaw; J. Lott; M. Breedy; L. Bunch; P. Feltovich; M. Johnson; H. Jung. 2008. New Developments in Ontology-Based Policy Management: Increasing the Practicality and Comprehensiveness of KAoS, IEEE Workshop on Policies for Distributed Systems and Networks, 145-152.

[23] Carminati, B.; E. Ferrari; and A. Perego, "Rule-based access control for social networks," in Proc. OTM 2006 Workshops, ser. LNCS, vol. 4278. Springer, Oct 2006, pp. 1734–1744.

[24] Masoumzadeh, Amirreza; James Joshi. 2010. "OSNAC: An Ontology-Based Access Control Model for Social Networking Systems." Social Computing (SocialCom), 2010 IEEE Second International Conference on Social Computing, 20-22 Aug. 2010, Minneapolis, MN, 751 – 759.

[25] Horrocks I.; Patel-Schneider, P.; Boley H.; Tabet, S.; Grosof, B.; Dean, M. 2004. "SWRL: A Semantic Web Rule Language Combining OWL and RuleML." /www.w3.org/Submission/SWRL/ .

[26] Samuel, K.; L. Obrst; S. Stoutenberg; K. Fox; P. Franklin; A. Johnson; K. Laskey; D. Nichols; S. Lopez; and J. Peterson. 2008. "Applying Prolog to Semantic Web Ontologies & Rules: Moving Toward Description Logic Programs." Journal of the Theory and Practice of Logic Programming (TPLP), M. Marchiori, ed., Cambridge University Press, Volume 8, Issue 03, May 2008, 301-322.

[27] Samuel, K.; L. Obrst. 2007. "Answer Set Programming: Final Report on a Comparison Between ASP and Prolog for Semantic Web Ontology and Rule Reasoning." October, 2007. MITRE MTR090069.

[28] Obrst, L; Stoutenburg, S; D. McCandless; D. Nichols; P. Franklin; M. Prausa; R. Sward. "Ontologies for Rapid Integration of Heterogeneous Data for Command, Control, & Intelligence." Chapter in: Obrst, Leo; Terry Janssen; Werner Ceusters, eds., 2010. Ontologies and Semantic Technologies for the Intelligence Community. Amsterdam, The Netherlands: IOS Press.

[29] McCandless, Dru; Leo Obrst. 2009. "Dynamic Web Service Chaining using OWL and a Theorem Prover." 3rd IEEE International Conference on Semantic Computing, Berkeley, CA, USA - September 14-16, 2009.

[30] Preuveneers, D.; Y. Berbers. 2008. "Encoding Semantic Awareness in Resource-Constrained Devices," IEE Intelligent Systems, March – April, 2008.