Cross-Boundary Policy Administration for an Investigator-Matching System

Arnon Rosenthal, Cleo Casipe, and Conrad Chang

Abstract—**This work addresses security policy administration, as motivated by a knowledge management application. We describe the policy approach used when reporting matches -- pairs of investigators from different agencies whose queries appear to be about similar topics. Release policies for queries, investigator information, and match results must reflect preferences of many stakeholders. At the same time, policy must be easy to administer -- the system will be rejected if each policy change requires professional administrators. To meet these needs, we propose to capture most of the policy specification as assertions of simple facts or situation derivations. Instead of resolving conflicts by global rules, each situation's or policy's administrator provides a derivation function to derive an unambiguous situation value or action.**

***Index Terms*—Policy administration, security policy, matching, distributed policy, OWL**

I. Introduction

MUCH government and industrial knowledge resides in people, not in databases or documents. While there are many tools for finding on-line information, sometimes it is more valuable to put two investigators in touch, so they can share ideas and concerns. We addressed this challenge, in settings where independent investigators from many organizations are querying the same relational database (e.g., of insurance claims, goods shipments or financial transactions). It appears possible to extend the approach to situations with multiple databases, by applying query and policy translation techniques.

Our prototype attempts to identify investigators who appear to be interested in the same data. Our *query matching* algorithm estimates similarity of each pair of queries, based on use of similar data values, plus estimates of how central each value is to the query's real topic. For example, two queries that both reference the SSN 123456789 are likely to be related; queries that reference similar names (John Smith, John S Smith) somewhat less so; queries that mention a giant entity (has a checking account at Citicorp) much less so. A match describes the two queries, their estimated similarity, the two investigators, and the nature of the connection. Matching based solely on query text is one end of a spectrum; techniques that look at query results or full database contents could produce more matches, but will be harder to deploy because they require more access to sensitive or proprietary information. We focus on policy administration; the system and matching algorithm will be described in a separate paper.

Potential participants are cautious about releasing matches, so release policies must be supported. Participants may wish to hide, filter (i.e., subset or anonymize), or urgently share match results, based on contents, the recipient, the investigation, etc. Release may be to a third party (e.g., a highly trusted manager). We use the term preference to denote a stakeholder's wish that the system does not guarantee to uphold. Policy refers to the (necessarily unique) treatment that the system actually enforces.

The policy will be derived by somehow blending preferences from many sources: all the investigators whose queries were used to generate a match (perhaps transitively), plus their agencies and managers. Even from one person, preferences may overlap, e.g., one preference for Shipment data and another for data about NYC. Since the participating agencies are independent, we cannot rely on a central, professional administrator to prioritize and arbitrate all their wishes, nor to translate the result into a formal policy language (e.g., XACML). Since the administration process must blend all these preferences into an unambiguous, executable policy, we need a process that works in our environment.

Ease of administration is our crucial goal. As software connectivity improves, we need to explicitly specify which of the new potential users should be allowed access. Administration has already become the greatest cost of ownership for databases (exceeding software and hardware). As enforcement techniques mature, we expect a similar problem for policy -- the people process of administration will become the major roadblock.

Most policy language research aims at expressive power, automated inference, and efficient enforcement. Enterprises' main long term risk, however, is that administration may be so awkward that users either specify inappropriate policies or simply refuse to participate. A model-centered analysis, or any other analysis that does not explicitly consider administration, cannot give us confidence.

This is a position paper, not a report of finished results or a detailed model. Our aim is to steer researchers toward treating administration as a critical challenge to be explicitly examined when proposing a new model. When addressing enterprises or multi-enterprise teams, they should consider the issues raise by scalability [Ro05]. In particular, we hope to encourage other researchers to shift attention from monolithic languages and after-the-fact conflict resolution to a composite approach

that, we believe, has a better chance of success.

II. User Requirements

This section discusses fundamental requirements for administering policy in enterprise and cross-enterprise systems. The requirements are so basic that they appear even in our small "matching" application. It is expressed in general because they illustrate many of the difficulties encountered in administering large scale enterprise and cross-enterprise polices.

Complex policies using complex formalisms are beyond the capacity of most of the people who know what is organizationally appropriate. Nevertheless, there are certain basic user requirements that need to be considered regardless of the complexity of the policies.

Requirement 1: Suit people of varying skill levels. Policy preferences need to be captured by nonprofessionals, who cannot use complex formalisms. Professionals are too costly and often are remote from actual business needs. On the business side, sometimes an expert understands only one aspect of the policy problem (e.g., security but not privacy law). Often, a local expert can provide *descriptive* information about a situation, but not enough to *prescribe* the *best* action.

In our application, stakeholders whose preferences should be considered include investigators, managers, power users, and professional administrators working for large organizational units. The investigators and local managers often know what is best, but may not bother to specify all details, so the professional's task includes stating basic safeguards (e.g., release only to Insurance companies).

Each user category requires a coherent model with constructs simple enough for them to understand. GUI builders cannot provide coherence as an add-on. Also, a tool suite is needed, e.g., for "can X happen" or "who can access Y" or "what if" analyses. We assume that the user-provided specifications are compiled to an enforcement-friendly form; the latter may, for example, combine predicates that the interface collects from different people.

Requirement 2: Support multiple stakeholders. Enable policy to be driven by multiple considerations. Considerations include security versus privacy versus utility, or investigator versus manager versus enterprise preferences. One person may manage several aspects, or they may be distributed to multiple experts.

The system must enable the preferences to be stated separately (Requirement 1) and edited separately (Requirement 4), and yet allow business (not technical) experts to specify a merge that is appropriate in the current situation. Global rules are unsuitable – "Deny wins" fails when a General says "Allow" and a Private says "Deny". We also want to manage a variety of actions (e.g., Allow, Notify security officer, Modify the request, Audit).

Requirement 3: Minimize duplication and inconsistency. Frequently, concepts reappear in many policies, e.g., emergency, two factor authentication, consultant relationships. Many of these concepts may also be defined outside the policy system, for ordinary application purposes. Unfortunately, today's systems often make it easier for administrators to write their own version rather than to reuse, or to reuse via cut-and-paste. (Upper management mandates for more structured but more time-consuming procedures are rarely effective.) One consequence is wasted work, in re-expressing similar situations. The consequences at the enterprise level can be equally serious--inconsistent treatments and no way to impose a change that affects all usage.

Requirement 4: Small changes should require only local editing. It should be possible to change a detail (what constitutes an emergency or a privacy rule) without reading (let along changing) large chunks of policy specification.

III. Design Approach

We aim to provide a layered formalism, where the simpler layers can have automated reasoning and be used by a much wider set of people. We aim to provide small units of work that can be independently reused, edited and delegated (thereby meeting several requirements). Finally, the person responsible for a unit (whether a whole policy, or a situation assessment within a policy) is responsible for providing an unambiguous response.

A. Layered formalisms

The more expressive the formalism (compare general programming to simple relational query interfaces), the more difficult it is to provide a good interface, and the more difficult to provide powerful analyses (e.g. "can X happen?").

The users' experience can be simplified by providing several formalisms, which we (over)simplify as layers. It is not sufficient to identify simple constructs within an existing formalism – they must provide a coherent conceptual view of some part of the system, and users' work in their formalism must fit into a coherent methodology. In such a system, the relative simplicity of each layer both aids users, and makes it possible to provide more powerful structuring and analysis tools.

Most users merely assert simple facts (attributes) that describe themselves, the controlled resource or the system context.

1) Power users define new *situation* classes, to represent more complex conditions, e.g., "is an emergency", "is a protected celebrity", or "has conflict of interest". A *situation's* value on a particular request is either asserted directly (e.g., the Michael Jackson court case is an Emergency), or derived from more primitive inputs by an administrator-defined formula. She may then delegate some of the inputs to other administrators. For example, an investigator, manager, and agency may each control one input.

2) Professional administrators define event/condition/action rules. Rules are harder to reuse, to combine unambiguously or to reason about. They have fewer semantic relationships; for example, types, value range constraints, and "IS-A" are poorly suited to rule sets. Fortunately, because situations have already been assessed, less needs to be managed at this layer.

3) The professionals are also responsible for combining multiple proposed rules that stakeholders may suggest. There is no general calculus of rule combination. Several techniques will be discussed in section IV-D (Combining Actions).

Steps 1 and 2 are *descriptive*, intended to describe the current state of the world rather than the best possible world. They are best seen as part of the enterprise's data, partly supplied for policy reasons, partly for other applications. Trust assertions may be attached (as in SAML). Cycles are not allowed. Note that there is no artificial boundary between policy-relevant assertions and general enterprise data. We are currently exploring sample policies, and gradually clarifying our formalization.

Rule definers, who prescribe behavior, need both technical skills (to handle the rule language), and business judgment (to determine the "best" tradeoff among security, privacy, and application utility).[1] Since global conflict resolution rules (e.g., Deny beats Allow) cannot be universally applied (e.g., when the CEO says Allow), we are exploring alternative administrative modes. Conflict avoidance may be possible by defining more nuanced situations that permit a clear judgment. These ideas are not new – we want to see if pushing complexity into situation definitions makes the remaining combinations relatively manageable. The idea is to decentralize and ask administrators to assess tradeoffs in a specific situation they understand, not a combination with "whatever policies determined to be relevant".

Policies (especially with actions beyond Allow/Deny) are difficult to combine and to analyze. Therefore, as much administration as possible should be done in terms of data assertions. Data formalisms enable typing, composition, query, and powerful automated tools.

B. Accumulate modular chunks of specifications

We describe a formalism approach to meet this goal, based on OWL. Further details appear in Section IV-A

We anticipate that an enterprise policy formalism and system will be a framework into which locally-preferred pieces are plugged. It is infeasible to get an entire enterprise to adopt (and stick to) detailed semantics on every issue. Thus, we specify only the minimum that we need, allowing detailed models to be plugged in. For example, we assume each privilege's administrator can delegate to others, but allow different privileges to extend these semantics differently (e.g., whether revoke cascades).

The natural starting point is to encompass the power of today's systems to describe attributes of users, controlled resources, requests, and information in ordinary databases (sometimes called *context*). The security community has produced attribute- and role-based access controls (*ABAC, RBAC*) to capture such information.

However, a much better choice is available. OWL, the Semantic Web Ontology Language, is far better at capturing knowledge and supporting reasoning than any widely used competitor in the security community sometimes used to describe general enterprise information of interest to policies (e.g., organization charts, personnel assignments).

OWL classes and assertions (*resources* of the form (resource, property, value) provide a unified treatment that can categorize and assert attributes and connections for any sort of information.

OWL was intended to enable machine reasoning, not to be friendly to administrators who maintain the knowledge. We therefore employ a tiny extra profile -- an insistence that each situation include a "meaning" property (in English). The meaning constitutes the contract that must be maintained as the system changes. (The role based methodologies we have seen do not formally emphasize such a contract. Consequently, one cannot say whether it is "right" to include a new user in a group or community, or a new privilege within a role.

C. Each object's administrator disambiguates it

Policies will be created top-down, though optionally reusing previously defined resources. Each event (i.e., request for some operation on an object) will be associated with *just one* policy. The administrator of each object's policy is responsible for giving it an unambiguous meaning; it is a good practice to define this meaning in terms of already-defined situations and actions.

First, one may define new, disjoint Situations, to select unambiguously among the current set of Actions. For example, one may partition based on an attribute (which state an investigator comes from), accumulate weights and set thresholds, or write general logic. Syntactic sugar can help, e.g., for declaring exceptions and inserting corresponding negatives into the derivation of other situations.

Alternatively, the administrator may create a composite action that did not exist before, and refine the situations to choose among this wider set. Section IV-D seeks to meet the objectives of all the actions being considered; other times, the administrator may simply impose a compromise.

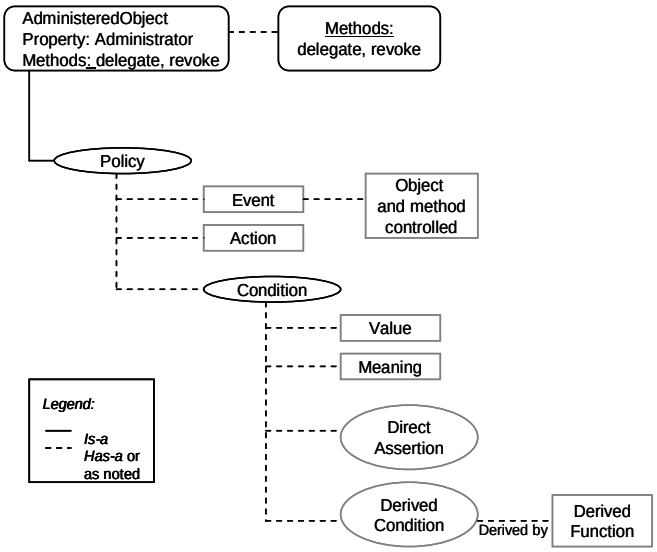
IV. Proposed Formalism and Methodology

A. Formalism

The formalism must identify *protected objects* (objects that have policies) and define classes of data assertions (*situations* or synonymously, *condition*), and policies that reference data assertions. All of these are *administered objects*, meaning that they possess an administrator who controls their definitions and (optionally) delegates to create other administrators. Both the ordinary system information being controlled (the policy targets), and the policy information are treated as OWL resources. Figure 1 shows the key concepts, but omits some of our details.

[1] In some organizations, the rule definers mainly translate high level guidance to the objects at hand. We consider that mainly a problem of semantic heterogeneity, and do not address it here.

Fig 1. Proposed Policy Schema (high-level).



Each policy is a set of Event/Condition/Action rules, in which the Conditions are logically disjoint (so the Action is unambiguous). A policy may also have annotations that help a human decide what to do in case of conflicts, e.g., pleas that it is important, or mentions that a weaker action might be acceptable.

Each condition is an OWL resource. Additionally, it has a meaning stated in English, so administrators can agree among themselves about what the assertions mean, and so management or auditors can review their work.

A condition class can be *direct* (an administrator sets the value) or derived (an administrator defines how it is derived from other resources). The derivation function can be a Boolean expression (describable as OWL union and intersection) or encapsulated (in any language). We want straightforward semantics, so we require that derivations form an acyclic graph.

The policy system is invoked to process an Event that describes a request to access a protected object, (e.g., Read contact info on Investigator $I; Release match $M). Each Event type has a unique policy associated with it. (Events and conditions each create a set of assertions. Subsequent steps may query these, as part of the request's context.)

Where two rules use different actions, the administrator must determine what to do. (When the use the same action with the same arguments, it is best to combine by defining a new condition via Boolean AND or OR of the original conditions).

To enable either sort of combination, the rules need additional information about action semantics and administrator's intent. We will we will assume that actions are partially ordered in a lattice, based on permissiveness (with "Allow" at the top). Administrators who provide a preferred rule must explain whether their specified action is the "most" or the "least" permissive they consider satisfactory. Semantic properties of actions need also to be tracked. Relevant properties include idempotence (i.e., repetition does not change the result), subsumption, and commutativity.

We provide a few operators that derive a single policy from a list of policies, i.e., syntactic sugar for situations where it is inconvenient to write a single ECA rule. We provide the following (each of which can be reduced to ECA rules by defining new conditions or actions):

- Default actions, if no conditions apply (one default condition per event type).
- XACML combiners (e.g., to choose the first policy whose condition holds, or "Deny wins", or to execute all actions in specified order.
- Combiners specific to the semantics of particular Actions, e.g., that realize Write Audit Record is independent of Deny,

B. Methodology

A process outside the policy system defines an event type and appoints a single subject (person or role) as administrator for it. Here is how the administrator proceeds to define policy.

1) Determine who are the stakeholders in this decision. For example, there may be a privacy concern, a corporate security concern, and a manager who understands who really needs to see this data.

2) Determine the set of actions they are recommending. Also, determine a default action if nobody expresses an opinion. Now, as discussed in section III.C, either uses situations to disambiguate or define a compromise action, or both.

3) A situation assigns a value, often Boolean, to describe some real world state. For each situation, the administrator must provide a *meaning, in human language* (for us, English), as a guide to herself, or whoever provides values for this situation on individual requests. Explicitness is also valuable for examining whether the treatment is still appropriate, and for explicitly changing the criteria.[2]

It is the administrator's responsibility to ensure an unambiguous value for the situation, evaluated on any request. She may assert the value directly (independent of a particular request), say that it will be derived from a lower level situation (possibly controlled by another administrator), or provide a derivation formula (Boolean, arithmetic, etc.). For finer discrimination, she can define subclasses.

- An input might refer to a condition someone else has already defined. (If so, we have achieved Reuse, a very good practice).[3]
- Alternatively, the administrator might define some additional conditions (e.g., Dire Emergency). She then

[2] Descriptions in terms of the external world are greatly preferred to prescriptions such as "If I say so". The descriptions should be in terms of something simpler. "If the good of the company warrants" gives no help. A high level statement "if benefits exceed costs" suggests doing something far too rarely done in security policy -- estimating benefits and costs.

[3] The reference must not create a cycle. The system could check, but since the administrator is seeking simpler concepts, we suspect that cycles will naturally be avoided.

drills down to each additional condition, which is handled just like the parent (i.e., by delegation, direct assertion, or derivation,

4) If the stakeholders' desired actions are more varied (e.g., Deny, Write Audit Record, and ChargeCustomerAccount), then the administrator (aided by the system) will need to:

- Create composite actions that weave these together in appropriate ways (sequentially or concurrently). Since she is the authority, she may also read the annotations and provide a compromise.
- Establish a condition for each composite action, keeping the conditions disjoint. Each condition is handled as in item 3.

C. Illustration of methodology

We consider a class of objects whose instances are distributed, namely insurance investigators. Creation is bottom up, so each participating company creates its own subclass of Investigator, and designates a policy administrator for it. We now follow the example for investigators at two insurance companies, "Crash" and "Lucky". Each controls the policy on releasing information about their own investigators, although they may consider urgency estimates from their outside partners.

The Crash administrator determines that "Crash" will support only two actions. Allow and Deny. She will allow three categories of stakeholder to describe their situation as a preference to be considered -- the investigators involved, his division manager, and the corporate policy board (on which she sits). She decides that each preference can be an integer expressing the urgency of releasing the protected object, negative for Deny. She sets range constraints on urgencies: the corporate decision can use any weight, the individual can set weights (-10, +5) and the division manager (-2, +5).

The top level derivation will sum all three situation estimates and Allow if the result is positive. Here, the corporate administrator writes all the derivation functions (although she might have delegated to division managers for intra-divisional sharing).

Now the top administrator delegates DivisionWeight and IndividualWeight to division managers and the investigators. Keeping responsibility for Corporate, she writes a policy (If DireEmergency then +150; else if recipient company is in friendly list then +3; if in unfriendly list then -4). In effect, she lets herself be outvoted by the manager and investigator, except in dire emergency. She defines DireEmergency as OR (US_Govt.EmergencyDecreed, CEO.TemperTantrum).

Note that the derivation here reuses conditions others have defined, rather than using her own. Finally, she creates a policy on the above inputs to DireEmergency, so that an update to either of them is placed in a list for daily review.

Meanwhile, the top administrator at Lucky Insurance indicates that three similar stakeholder groups will express preferences, but they may do it by defining full policies (giving them more power, but more complexity). He defines some situation (classes) that all groups can reuse, and defines the Actions that may be used in policies, as a lattice. Each preference will be in the form of "at most this permissive" Action. He then states that the default policy is to invoke the greatest lower bound of all the desired actions, but that if policies are annotated "appeal to the administrator" he will use his best judgment. (This customized treatment provides flexibility; its downside is that the other stakeholders do not know if their specified upper bound will be enforced.)

D. Combining actions

Each event must have a unique policy, and hence a unique action. Actions (i.e., obligations) go beyond Accept/Deny, to include auditing, payment, notification, etc. An automated system can try to create a composite action that achieves all the individual objectives. A policy definer can use the combination of actions from a set of "at most" preferences [or a set of "at least" preferences], by taking the greatest lower bound [or least upper bound].

The nature of the upper bound depends on the operation. We illustrate for "least permissive" semantics. Allow and Deny resolve to Deny. "Allow unconditionally", and "Allow if Audit record X is written" resolve to the latter. "Allow if audit record X is written" and "Allow if audit record Y is written" resolve to "Allow if audit record with information $X \cup Y$ is written. (Union semantics need to be defined, e.g., union of sets of assertions, or concatenation of text).

Suppose, for example, that the division manager says "don't care" (i.e., Allow if others concur), the Investigator says "OK to release anonymous email address to anyone" and the top administrator says "OK to release to any Investigator in a company on the Friendly list". The effect is to allow release of anonymous email addresses to people at friendly companies.,

V. Related Work

There is a large literature of policy models, and a very much smaller one on administration methodologies for employing those models. Detailed arguments that are orthogonal to our concern about splitting administrative work will not be surveyed here.

Originally, privileges were granted one at a time. Role based access control aimed was a next step. It was quite successful in reducing the volume of administrative work by enabling grants to sets of users or privileges. The NIST standard [FKC] is very simple structurally, optimized for easy enforcement. Unfortunately, it is less natural for administration than its predecessor, RBAC96 [San97], which helps administrators define meaningful units by having separate constructs for privilege sets (called roles) and groups. For both of them, though, great skill is needed to define the role set, e.g., to manage both Dept. Head, and Audit Dept. Head (which carries additional privileges). Extensions have been proposed that add predicates, obligations, and subtle controls over administration. These greatly increase expressive power, but are more difficult for users and tools. Lack of a standard for them reduces enterprise interest.

Attribute Based Access Control (ABAC) avoids the need to

artificially map the entire enterprise to a single untyped graph, by instead allowing predicates on those natural concepts. Researchers have shown that a specialized logic can formalize this well [WWJ]. However, using a rich logic known to a tiny community inhibits researchers' communication, and lack of a tool suite reduces its practical attractiveness. Others (e.g., [DD+]) used highly expressive languages that resist reasoning and are suited only to programmers.

Several industry standards (XACML, WS-Sec) are geared to ABAC, enabling predicates over attributes to be expressed straightforwardly. However, they are still confined to the security community, so the skill base and tool suites do not match semantic web languages. Obligations (Actions) are supported. However, these languages are geared toward execution rather than administration (a natural choice, since without an execution engine, policies have little use), and do not seem to provide a simple sublanguage for nonprogrammers and for treating sub expressions as intermediate, reusable concepts. The four-valued logic is at once awkward for reasoners.

Researchers have aimed at securing the semantic web [QA, TB+, KB+]. Their work uses OWL's inference to simplify administration (e.g., automatically applying a general policy to subtypes). For simple cases, it provides a strong basis. Nevertheless, complicated constructs (negative permissions, obligations) make it harder to see the justification for the semantics. General expressions and non-binary predicates are opaque to OWL.

Many of the existing formalizations assume that the different preferences are expressed independently, and then a global rule (e.g., Deny beats Allow) arbitrates. This treatment is easy for enforcement, but hard for administrators, due to its peculiar authority structure – whoever has the power to state the global rule has great responsibility and workload. With the schemes we have seen, there are three alternatives, none perfect.

- *Have the conflicting participants negotiate directly, so that one changes their policy.* There is strong opportunity for a stalemate -- neither party has much incentive to give in. An arbitrator may be needed.
- *Attach exceptions at the arbitration layer.* The arbitration process becomes opaque, and thus participants cannot easily tell if their preferences (proposed policy) will apply.
- *Provide weights, and let the strongest preference wins.* We have seen both "strong/weak" (i.e., two level) and numeric weights (as at Crash Insurance). Such systems could provide a useful increase in power within a bounded setting, but will have difficulty scaling, requiring everyone to use the same urgency scale.

VI. Summary And Future Work

This position paper argued that *administration*, not enforcement, is the central problem for a policy system. An application involving matching of investigator interests was used to illustrate the critical issues in scaling to enterprise systems that the literature does not address head on.

We then sketched out an approach that is robust against these unavoidable difficulties. Principles include:

1*) Split the formalism, and the policy administration products, into parts.* Let each part be as simple as possible for the intended task, helping naïve users and also tool-builders. Emphasize data assertions, which have clear meaning (is Smith a detective?), and have excellent organizing and storage technologies (ontologies, databases).

2) *Provide tools for making change easy, and for analysis.* Administrators will migrate to a new approach only if it is easier for them. Change and analysis seem to be low hanging fruit.

3) *Operate with both top down and bottom up*. Top down gives a clear responsibility for avoiding ambiguity, and clear delegation. Bottom up provides reusable resources. The main challenge is to make this easy.

4) *Where whole policies must be combined, provide automated help.* Help in creating disjoint conditions, in combining actions according to a lattice, and in understanding the guarantees (or lack thereof) that each stakeholder gets.

Obviously, our formalism and methodology are incomplete and untested. So the obvious next step, for us or other researchers, is to test whether such approaches really do simplify administration.

A second major direction is to deal systematically with the semantic heterogeneity between the objects referenced in high level guidance documents, and the objects to which policies are actually applied. By combining with an ontology-based data integration environment and extending the work in [QA], we believe that the effort could be reduced, traceability improved, and useful semantic knowledge captured on the fly.

References

[RS00] A. Rosenthal, E. Sciore, "View Security as the Basis for Data Warehouse Security", CAiSE Workshop on Design and Management of Data Warehouses, 2000.

[FKC] D Ferraiolo, R. Kuhn, R. Chandramouli, Role Based Access Control, Artech House, 2004

[San97] Ravi Sandhu. Rationale for the RBAC96 family of access control models. Workshop on Role-Based Access Control. ACM, 1997.

[DD+1. N. Damianou,N. Dulay, E. Lupu, M. Sloman: The Ponder Policy Specification Language. In proceedings of Workshop on Policies for Distributed Systems and Networks (POLICY 2001)

[Ro05] A. Rosenthal, "Scalable Access Policy Administration: Opinions and a Research Agenda", IFIP 11.1 and 11.5 joint working conference on Security Management, Integrity, and Internal Control in Information Systems, 2005.

[WWJ] Lingyu Wang, Duminda Wijesekera, Sushil Jajodia, "A logic-based framework for attribute based access control", Proc. 2nd ACM Workshop on Formal Methods in Security Engineering (FMSE 2004)

[TB+] G. Tonti1, J. Bradshaw, R. Jeffers, R. Montanari, N. Suri, and Andrzej Uszok "Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei, and Ponder", Semantic Web Conference (ISWC), 2003. Springer.

[KB+] L. Kagal, T. Berners-Lee, D. Connolly, D. Weitzner: Using Semantic Web Technologies for Policy Management on the Web. AAAI