A Model of Triangulating Environments for Policy Authoring

Kathi Fisler
WPI Computer Science, USA
kfisler@cs.wpi.edu

Shriram Krishnamurthi
Brown University Computer Science, USA
sk@cs.brown.edu

ABSTRACT

Policy authors typically reconcile several different mental models and goals, such as enabling collaboration, securing information, and conveying trust in colleagues. The data underlying these models, such as which roles are more trusted than others, isn't generally used to define policy rules. As a result, policy-management environments don't gather this information; in turn, they fail to exploit it to help users check policy decisions against their multiple perspectives. We present a model of *triangulating* authoring environments that capture the data underlying these different perspectives, and iteratively sanity-check policy decisions against this information while editing. We also present a tool that consumes instances of the model and automatically generates prototype authoring tools for the described domain.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications

General Terms

Design, Security

Keywords

Policy authoring, Authoring environments

1. INTRODUCTION

Authoring access-control policies is difficult. First, as with any configuration problem, the author has to wrestle with a large state space due to the number of combinations of policy elements (such as roles or resources). Second, people make policy decisions based on different, sometimes conflicting, viewpoints (e.g., putting social interactions first versus putting security first). Third, there are often subtle overlaps between principals (e.g., a person may fulfill multiple roles), resources (one may be part of another), actions, etc., which make it hard to catch all situations and thus to get the policy right. Fourth, the authors are often non-technical (e.g., in a company, they may be business or marketing staff), who may lack familiarity with software security considerations. Finally, the consequences of a mistake can be enormous, especially if it results in leakage of sensitive data. For all these reasons, policy authors require significant help fleshing out their requirements, expressing them in policy languages, checking for subtle corner-cases, and identifying potential conflicts.

State-of-the-art policy-authoring tools have taken several steps to address these issues. Analyses that detect inconsistencies or incompleteness within policies abound [2, 14, 17]. Visualization techniques attempt to provide a more holistic view of policy matrices [23]. Natural language interfaces provide a semi-formal method for stating policy rules [3].

Despite their strengths, these advances stay focused within the space of the policy being authored, and do not take into account the broader social context in which the policies are written. People base policy settings on information that is not formally part of the request, such as trust or privacy concerns. As these concepts often do not manifest formally in the rules, many policy logics and authoring tools neither capture nor exploit them. Instead, people must remember to manually check that policies conform to their personal, semi-formal metrics. In addition, people often have multiple mental models against which to make policy decisions. This complexity, combined with the subtle relationships between roles or resources, means checking these constraints manually is untenable for all but the smallest of policies.

We need more nuanced models of policy authoring to address these problems. In particular, models need to more fully embrace the role of "sanity checks" that help authors triangulate policy decisions against auxiliary information. As triangulation is an ongoing, rather than posteriori, process, models should also embrace the iterative nature of authoring. This allows us to insert analyses and sanity checks as the policy evolves; this can help a user who makes a decision based on one mental model without realizing its impact on another. Embracing sanity checks brings a key component of authoring practice into tools.

The main contribution of this paper is a model of policy authoring with two novel features. First, our model incorporates queries that triangulate policies against different perspectives. Second, it captures the iterative development of policies, integrating formal problem detection and resolution with warnings. A secondary contribution is a prototype tool that demonstrates the utility of the model: given an instance of the model, it generates an interactive policy-authoring environment that prompts users as they incrementally author

SACMAT'10, June 9–11, 2010, Pittsburgh, Pennsylvania, USA.

a policy. We illustrate the model and the tool using two examples: faculty-hiring software and a social network.

2. TRIANGULATION IN AUTHORING

The idea that people view security decisions through multiple lenses—some of them informal, many of them social, and some of them conflicting—is well established in the literature. Palen and Dourish [22] characterize privacy management as a dynamic process of setting boundaries. In particular, they identify three broad boundaries that appear in privacy practice: disclosure, identity, and temporality. They describe privacy management as "something of a balancing act, a resolution of tensions not just between people but also between their internal conflicting requirements". Nissenbaum's work [19] on trust as a metric for security in software and computing technologies identifies many nuances through which people express trust. A semi-formal, user-specific interpretation of trust seems best able to accommodate these subtleties. Lederer, et al.'s guidelines for interface design for privacy-related access-controls [16] note that such systems effectively integrate at least two mental models for users: one of system operation and one of identity portrayal and disclosure. Supporting multiple mental models during authoring in a flexible manner is one of our goals for triangulating authoring environments.

The importance of social considerations as a part of policy setting is also well established. In discussing models of security that account for social dimensions, Dourish and Anderson [6] note that "privacy is not simply about how information is managed, but about how social relations are managed". While access-control is not the same as privacy management, the two inexorably overlap, particularly through social networks and related applications. Social relations matter even in more conventional access-control domains such as file sharing. In a user-study, Whalen, Smetters, and Churchill [31] observed many cases in which people set sharing permissions based on trust that another party would use granted permissions appropriately. They also noted that the distinctions underlying social attributes are not always clear-cut, such as who should be considered an "insider" versus an "outsider" when collaboration occurs across institutions. In summarizing the range of perspectives they observed, they note that

> our interviews revealed a wide range of "mental models" or belief systems around digital content protection and a concomitant range of practices. We were mildly surprised about this range within the bounds of one relatively small organization.

They further call for authoring methods that take these methods, and their nuanced social dimensions, seriously:

> there have been no systematic user studies of the basic access control models deployed in the vast majority of current systems—or, for that matter, the social control models that people are tacitly trying to apply in their particular activity contexts when they use these security mechanisms. In systems, the implemented models generally have tremendous expressive power, potentially leaving users awash in a sea of very fine-grained access control settings.

Our vision of triangulating authoring environments provides a framework for formally handling social controls, though as way to sanity check, rather than define, policies.

Beyond these papers, we witnessed triangulation in policy authoring first-hand while developing a software application to support faculty hiring for a US-based academic department. As part of this process, we discussed access-control requirements with several faculty in the client department. These faculty frequently checked their evolving requirements against subjective relationships among data. For example, some faculty discussed how their access decisions seemed to contradict the reporting hierarchy among the staff, questioning whether it was reasonable that someone with more responsibility would have fewer permissions. These faculty weren't convinced that this was a problem; however, they did want to review any decisions that violated the reporting hierarchy as a sanity check on the policy. Similar examples arose several times: different faculty referenced the relative loyalty of different roles to the department, relative trust in different roles, and analogies between the sensitivity of job application components and that of other information handled by departmental staff. Representative comments along these lines include:

- "We have grad students, who shouldn't get to see much, but then there are the grad students on the committee, who we trust more than the graduate students as a whole."
- "This department has always trusted its staff. Our decisions should reflect that. Faculty get to see everything, so perhaps the staff should too."
- "An application consists of statements, vitae, and letters. The letters are the most sensitive and should be viewable only by faculty."

It is worth noting that the values of attributes such as trust and sensitivity are somewhat subjective in this context (whereas they might be objective in a MAC-style policy): different faculty placed different trust on various roles within the department. This argues for treating information used for triangulation separately from objective attributes on which actual policy rules are defined. Our model will adopt this distinction, as shown in section 3.1.

3. A MODEL OF TRIANGULATING AUTHORING ENVIRONMENTS

The authoring model we present embraces both the iterative nature of policy development and authors' reliance on sanity checks between policy decisions and auxiliary information. Policy authoring is naturally iterative because each action a user takes through an authoring tool results in a revised (often another partial) policy. Furthermore, in many settings, policy authoring is never really finished. For instance, in a social network, users add decisions for specific tuples ("allow friends to tag my photos"), introduce new data that is used to calculate decisions ("make Mary a friend"), create new roles ("create a tag for my professional colleagues"), and so on, always adding, deleting, and revising; a decision made today may impact data uploaded many years ago (and perhaps even long since forgotten).

We believe there are at least three kinds of analyses that provide useful feedback during authoring: flagging potential

problems as they are introduced, helping authors understand the impact of edits as they make them (such as a role overlap causing permissions to leak from one role to another), and reminding authors of issues they will need to address to complete a policy based on an action. Of course, the design of a tool that actually provides this feedback raises various user-interface issues, as we discuss in section 6.1.

The nature of triangulating sanity checks and their underlying data depends on the scenario for which the policy is being defined, and thus varies widely. Examples include:

- in software governing a job search, checking whether more-trusted roles have access to more parts of a job application than less-trusted roles;
- in a social network, confirming that sensitive personal details are restricted from professional contacts; or,
- in a firewall policy, checking when IP addresses in the same subnet do not receive similar permissions.

Such checks are interesting because policies that violate them are not necessarily incorrect: a social-network user might want to share intimate details with a professional colleague who is also a close friend. These checks thus serve as *warnings* to authors, rather than errors; they only indicate that the policy contradicts normal expectations. In contrast, other checks do unambiguously identify erroneous policies: for instance, in access-control languages it is typically unacceptable for a given request to map to both "permit" and "deny". Our model thus draws a distinction between problems and warnings.

3.1 Formalizing Authoring

First, our model must capture the shape of a policy: what request tuples look like and which outputs are available. The following definition captures these features. It is intentionally agnostic on the policy language used to capture rules.

DEFINITION 1. *A* policy space *is defined by*

- *A finite set* $D_1, \ldots, D_d$ *of domains.*
- *A tuple signature* $S = D_{s1} \times \ldots \times D_{sk}$*, where each* D_{sj} *is a domain from* $\{D_1, \ldots, D_d\}$*. A* request tuple *is any tuple* $\langle t_1, \ldots, t_k \rangle$ *in which each* t_i *is from the corresponding domain* D_{si}*.*
- *A set* $O_1, \ldots, O_o$ *of outputs.*

For a standard XACML- or RBAC-style policy, the domains are role, action, and resource. The tuple signature is role × action × resource. The outputs are permit and deny.

Domains are defined separately from request-tuple signatures for two reasons: some policies use the same domain in multiple positions of a request (such as the source and destination IP addresses in a firewall policy), and some policies draw on environmental information beyond what appears in the request tuple (such as the time or availability of a resource). Definition 2 adds the infrastructure needed to use the latter to specify policy decisions.

A policy space is merely a type signature for a policy. At a minimum, an actual, concrete policy needs to populate the domains and map request tuples to outputs. Our model also assumes that policy decisions may be based on attributes of and relations between domain elements. We capture policy decisions through rules in relational logic, written in a Datalog-like syntax; the model would support other formalisms without impact on the results of the paper. These rules induce a mapping from request tuples to (possibly empty) sets of outputs.

We augment these conventional policy components with three new concepts: a set of *problem queries* over the domains, outputs, and relations indicating the criteria for a finished policy; an additional set of relations on domain elements used for triangulation data; and a set of *warning queries* constituting sanity checks that reference the triangulation relations. One problem query might detect conflicting decisions through the relational algebra expression

permit(S,A,R) and deny(S,A,R)

while another mandates a decision for every tuple:

permit(S,A,R) or deny(S,A,R)

The latter may not apply to languages like XACML [11], which permit "not applicable" as a decision.

To triangulate against relative trust between roles, we might introduce a relation moreTrusted on role × role and define the warning query:

moreTrusted(S1,S2) → (permit(S2,A,R) → permit(S1,A,R))

This query records the expectation that if S1 is trusted more than S2, then S1 should have more permissions than S2.

All of these components together comprise an instance of a policy space that can be evaluated during authoring. Formally:

DEFINITION 2. *An* instance *of policy space* P *consists of*

- *a set of elements populating each domain in* P*,*
- *A set of relations* $R = R_1, \ldots, R_r$*, each with a signature over the domains of* P*.*
- *tuples populating each relation in* R*,*
- *A set of* rule forms *"*$B \rightarrow O(\bar{t})$*" in which* O *is an output in* P*,* $\bar{t}$ *is a request tuple in* P*, and* B *is a relational expression over the domains of* P*, outputs of* P*, and relations in* R*.*
- *A set of* problem queries*, defined as relational algebra expressions over relations for the domains and outputs, as well as those in* R*.*
- *A set of* triangulating relations $T = T_1, \ldots, T_c$*, each with a signature over the domains of* P*.*
- *tuples populating each relation in* T*,*
- *A set of* warning queries*, defined as relational algebra expressions over the domains of* P*, outputs of* P*, and relations in* R *and* T*.*

The separation of R and T distinguishes objective attributes used to define rules (R) from subjective attributes used for triangulation (T). We implicitly define a relation over the tuple signature for each policy output, such as permit(S,A,R). The relational algebra expressions we admit supports the usual logical connectives (and, or, not, implication) over ground terms of the form $rl(a_1, \ldots, a_j)$, where rl is a relation and $a_1, \ldots, a_j$ match the signature of rl. Each a_i that is not a specific element of the corresponding domain (such as S and R in permit(S,read,R), assuming read is an action)

is treated as an existentially-quantified variable around the smallest expression containing all occurrences of it.

Instances arise from successive edits to the components of a policy. Which operations make sense between instances, however, varies with the domain. Our definitions so far blur the critical distinction of which elements of an instance should be editable during authoring. For example:

- A medical records application may fix roles (primary physician, specialists, spouse) and resources (e.g., prescriptions, blood type) and allow users to customize only the decisions for each combination.
- A social network might fix the resources (photos, personal info, news feeds) but allow users to define private classes of friends (college, professional, neighbors, etc.) and the decisions for each combination.
- A small group trying to figure out their access-control requirements might want each user to specify their own roles, auxiliary relations, and triangulation checks as part of the authoring process.

To differentiate between these scenarios, a model of authoring requires a specification of which components of instances may be edited during authoring and what kind of edits are allowed, as well as any initial values of components. The next definition formalizes this.

DEFINITION 3. *The* components *of a policy space P consist of each domain in P, the set R, the set T, each relation in R and T, and the sets of rule forms, problem queries, and warning queries in P. An* authoring task *for P consists of a tuple $\langle init, actions\rangle$ for each component of P, where* init *is a set of initial values for the corresponding component and* actions *is a possibly empty subset of* {create, delete, delete-non-init}.[1]

In the action specifications, an empty set would fix the value of a component across all instances. The *delete* action differs from *delete-non-init* in allowing any element of a component to be deleted, rather than only those added since the initial instance. Different action settings for each policy component distinguish the authoring scenarios described above.

Why are problem queries editable, given that they define what makes for a completed policy? First, the policy author may be the same person writing the system that will use the policy. This person could plausibly need to adjust the problem queries during authoring. Second, a policy author may want to place additional restrictions on their policies. Given that problem queries are checked automatically during our model of authoring, allowing new problem queries extends the benefits of problem query support to these additional restrictions. The third, and most interesting, reason is because some policy authoring tasks occur over multiple stages, each handled by a different person. For example, in a system for managing conference papers (a domain for which we have authored production software):

1. The primary software developers would fix the request shape, valid outputs, and problem checks for access-control policies for conference management. They then pass along the partially-instantiated policy to:
2. The program chair, who specifies which roles and resources the particular conference will employ (subreviewer, short papers, demos, etc.). The chair might also stipulate additional rules, such as a conflict-of-interest policy, again using the environment to ensure that this does not interact strangely with the rest of the configuration.[2] This is finally handed to:
3. The system administrator, who might fine-tune some of the parameters, checking that these settings do not change the policy in an adverse way, before deploying it on the server.

The author in each stage would benefit from authoring tool support of the form described in this paper, perhaps with different triangulation data or problem queries per stage. Supporting multi-stage authoring tasks therefore suggests a hierarchical model of a sequence of authoring tasks, each expanding into a sequence of instances which in turn initialize later authoring tasks. As there are no general constraints on defining new authoring tasks from existing instances, we do not provide a formal model of sequences of authoring tasks.

Definition 3 captures the sequence of instances that comprise authoring, but does not utilize the intermediate instances in any meaningful way. The last component of our model integrates analysis into the authoring process, because the problem and warning queries naturally suggest running each query and giving the author information to help align the policy with any failed queries. But such analyses are only useful to the extent that good problem or warning queries have been defined! What if they don't exist?

Fortunately, the mere act of editing generates multiple instances of an evolving policy, which enables a query-free form of analysis based on differencing policies. Policy differencing [9] computes the set of requests whose outputs differ between two policies. Setting a decision for a single tuple may affect many other requests; once authors can express an analogy between two roles, for example, altering the permissions for one might implicitly alter those of another. Unless the end-user is able to keep all of the connections between output decisions in their head, some edits will have unexpected consequences. Differencing computes these consequences automatically, so the author can confirm that they were intentional, or at least desirable.

DEFINITION 4. *A* triangulating authoring environment *for an authoring task*

1. *Starts with an instance populated according to the* init *components of the task,*
2. *allows authors to edit policy components as specified in the* action *components of the task,*
3. *at each iteration, alerts the user to violations of problem and warning queries within the current instance, and*
4. *at each iteration, summarizes request tuples whose decisions changed from the previous iteration.*

There are, of course, many possible implementations of these ideas. Section 4 provides one concrete example.

[1] We eliminate *read* and *update* from the possible actions as read is necessary in order to define policies and the distinction between update and a sequence of deletion and creation isn't relevant in this context.

[2] The second author once saw a program committee meeting deadlock due to such a configuration problem, until a PC-member stepped in to masquerade as a chair.

Domains: role, resource, action
Tuple shape: <role, action, resource>
Outputs: permit, deny

Relations: analogous : **signature** <role, role>; **properties** {transitive symmetric}; **UI** ["role1", "role2"]
trust: **signature** <role, role>; **properties** {transitive acyclic}; **UI** ["More trusted role", "Less trusted role"]
sensitivity: **signature** <resource, resource>; **properties** {transitive acyclic}; **UI** ["More sensitive", "Less sensitive"]

Rule Forms: (permit S2 A R) and (analogous S S2) → (permit S A R)
(deny S2 A R) and (analogous S S2) → (deny S A R)

Problems: "decision conflict" = (permit S A R) and (deny S A R)

Warnings: "trust conflict" = (trust S1 S2) and (not ((permit S1 A R) and (not (permit S2 A R))))

Task Actions: *create* on {role, action, resource, analogous, trust, sensitivity, ruleforms}

Figure 1: The authoring-task specification for hiring policies.

3.2 Examples

To illustrate and validate our model, we use it to capture two different kinds of authoring tasks. The first defines access-control policies for faculty-hiring software (as discussed in section 2). The second is for a generic, tag-based social network.

3.2.1 Faculty-Hiring Software Configuration

Consider a software application for managing applications for faculty positions. In such systems, applicants submit their materials (vita, statements, etc.) via a Web interface. The software emails a letter-submission URL to each reference letter writer. Department faculty, and perhaps graduate students, can view and comment on the applications through the software. Administrative staff handle various requests from members of the department. Technical support staff maintain the infrastructure. This domain offers an interesting case study for access-control authoring because several requests lack obvious decisions: Should applicants be allowed to check which of their reference letters have arrived (and when)? Should students in the department know who has applied? Should administrative staff be able to read the reference letters? Should untenured faculty be permitted to read reference letters on senior applicants? As described in section 2, we observed several faculty use triangulation to help weigh these decisions.

Figure 1 shows an authoring-task specification for this domain. The first three lines define the policy space (definition 1). The "Relations" section defines three relations: one capturing analogous roles, one capturing relative trust between roles, and one capturing relative sensitivity between resources. The **UI** component of the relation definitions is for the environment generator described in section 4; the **properties** specifications are also for the tool, which automatically inserts tuples to maintain the stated properties as authors populate relations. This format does not distinguish between R and T explicitly. Rather, we infer this distinction from use: relations used in rule forms or problem queries lie in R; the rest lie in T. By this characterization, analogous lies in R, while trust and sensitivity lie in T.

Two rule forms derive permissions based on the analogous relation. In our experience, authors frequently express desired policies based on analogies between roles that they override in a few exceptional cases. While this is not a feature of many policy logics, we include it as it illustrates how exploring interactive authoring models drives research questions into policy logics (section 6.2 discusses this further). The single problem query captures the usual definition of multiple outputs for the same request tuple. The single warning query shown triangulates the policy decisions against the trust relation by flagging any pair of roles for which the less-trusted role has at least the same permissions as the more-trusted role. Many other warning queries could be defined relative to information sensitivity; we elide these as they do not illustrate additional concepts.

3.2.2 Social-Network Configuration

Social networks illustrate multi-stage authoring tasks. The first stage is for a policy internal to the social network that governs system-wide operations such as creating accounts and uploading content. The specification on the left in figure 2 captures the authoring task for this policy: through the resulting interface, the system's creators might permit only administrators to create accounts, and only members to upload, post, or tag items. The error criterion says that decisions must be unambiguous. This is similar in spirit to many other commercial access-configuration tasks.

The second, more interesting, form of authoring is performed by end-users, i.e., the members of the social network. A specification for this—which we explain below—appears on the right in figure 2. In principle, the interface generated by this model would constitute the "privacy setting configuration" page of the social networking site.

The second task specification considers an interesting form of sharing based on *tags*, as used on many Internet sites. It pre-defines a few tags, but end-users can use the generated interface to add tags of their own. Thus they assign people to member tags (e.g., "friends", "family", "professional contacts"), ascribe item tags to data (e.g., "work document", "party photo"), and specify sharing by permitting or denying all members of a member tag access to all data with an item tag (e.g., "permit friends to view party photos", "permit family to view family photos", "deny professional contacts to view party photos"). We have configured requests to be in terms of member- and item-tags, rather than concrete data, though this is obviously easy to change and a function of what the actual social network software does.

The first warning fires when a person is granted access based on one tag but denied it based on another. We treat this as a warning rather than an error because this situa-

tion is likely to occur with some frequency, and given this contradictory decision, the social network presumably denies access, so the user only needs to worry about overriding it when they want to permit access (though just being told about this overlap may give them unexpected insight into their social circles, causing them to reconfigure access). The second warning is even more interesting: it warns when a person can view a photo through a tag when an earlier friend request from them had been denied. These sorts of checks are very useful in social networks, yet we are not aware of any that offer them.

4. TOOL SUPPORT: AN ENVIRONMENT GENERATOR

We now describe QnA, a tool for generating triangulating environments for policy authoring. QnA consumes an authoring task (definition 3 augmented with a bit of user-interface configuration information, as shown in figure 1). From these, it automatically generates an interactive application that enables users to incrementally define policies while querying them about errors and warnings. A screenshot of the environment resulting from figure 1 appears in figure 3. The rest of this section refers to these figures as we discuss how QnA implements each aspect of our model of triangulating authoring environments (definition 4). QnA is built in PLT Scheme [7], using the PLT Web server [15].

Refining instances.

QnA currently supports a fixed configuration of actions on components of an authoring task. Specifically, it supports *create* on policy-space domains, *create* on tuples within relations in R and T, and a limited *create* action on rule forms that maps a single request tuple to a decision (as opposed to arbitrary rules). The sets of known relations (in either R or T) and the sets of problem and warning queries are fixed to their initial values. QnA supports this fixed configuration by automatically generating the following web-based forms (examples refer to figure 3):

- For each domain in P, a form for users to add elements to the domain (e.g., the "Enter a new role" option).
- For each relation in R and T, a form for users to add tuples to the relation (e.g., the "Specify trust ordering" option). QnA does not currently display the distinction between relations in R and those in T. As users change the contents of domains, the drop-down menus for specifying relations change accordingly.
- A form for users to add specific request tuples to the output relations (e.g., the "Specify Decisions" option).

Supporting other actions is simply a matter of figuring out appropriate interfaces for them (such as an interface for writing rule forms and queries). There is no technical difficulty with supporting a richer set of configurations.

Running queries.

Following definition 4, each problem and warning query must be evaluated against each instance as the policy evolves. The semantics of queries are those of relational algebra, assuming that the domains are closed for each instance. Any implementation of relational algebra should suffice. Our implementation uses Datalog: QnA maps domain and relation contents to facts against which it evaluates queries and computes outputs.

Our mapping of instance data to facts is standard and straightforward: for each tuple $\langle \mathsf{t1}, \mathsf{t2}, \ldots \rangle$ in relation RL, QnA generates the Datalog fact

RL(t1,t2,...).

For example, if $\langle student, studentRep \rangle$ is in the analogous relation, we generate analogous(student, studentRep).

Query evaluation is complicated by the presence of negation in our language of relational queries. Our query examples throughout the paper have included cases that (implicitly or explicitly) use negation. The "trust conflict" warning query in figure 1 shows an example that uses negation on both ground terms and conjunctive terms. While some Datalog implementations support the former, the latter is not directly supported in the Datalog packages we considered for our implementation. We have therefore built our own query evaluator on top of the Datalog package for PLT Scheme. Our evaluator assumes all queries use only conjunction and negation operators (a straightforward rewrite if necessary). For any negated query term not T, our evaluator implements negation via a closed world assumption [25]: it computes the set of tuples that satisfy T, complements this set relative to the set of all tuples over the domains as defined in the current instance, generates a fresh relation name and a fact asserting that name over each tuple in the complement, and replaces the original term not T with the new relation name. Positive ground terms and conjunctions with no nested negation terms are passed to Datalog directly in the usual fashion.

QnA generates warning and problem alerts for every tuple that satisfies the corresponding query. Figure 3 shows a warning arising from the "trust conflict" query: although faculty have been deemed more trusted than students, the current instance yields no permission that faculty have and students lack (as expected by the query). Since this is possible, though unlikely, it is a warning and not an error.

Resolving problems.

QnA evaluates problem queries in the same way as it does warning queries. Rather than just present the tuples that reflect a problem, QnA also asks the user how they would like to resolve the problem. Although QnA could do the same for warnings, our current system limits this feature to problems as they *must* be resolved whereas warnings need not be. In addition, warnings arise frequently, especially when a policy is only partially completed (the warnings in figure 3 reflect that the author has not begun to specify decisions for faculty); suppressing warning resolution may reduce information overload, as we discuss in section 6.1.

For each parameter binding that satisfies a problem query, we compute the facts that make the query true under that binding. We then present the user with options to delete some of the supporting facts. If the supporting facts include conflicting outputs on the same request tuple, QnA also presents options to override one conflicting output with the other. Figure 3 shows an example in the "Problems" box: the input request for a student to view comments yields conflicting outputs. This conflict arises from an analogy between student and studentRep that should generally hold, but not in the specific case of comment viewing.

QnA currently implements overriding through low-level manipulation of the relational algebra expressions that define rules and queries. If the author chooses to override one

Domains: role **initvals** {admin member},
action **initvals** {create post upload tag},
resource **initvals** {account item}
Tuple shape: <role, action, resource>
Outputs: permit, deny

Relations: owner : **signature** <item, member>;
UI ["Item", "Member"]

Problems: "decision missing" =
not ((permit S A R) or (deny S A R))

Task Actions: *create* on {ruleforms}

Domains: member, item,
action **initvals** {view comment search},
item-tag **initvals** {profile news photo},
member-tag **initvals** {friend friend-of-friend declined-friend-request}
Tuple shape: <member-tag, action, item-tag>
Outputs: permit, deny

Relations: mtags : **signature** <member, member-tag>; **UI** ["member", "tag"]
itags : **signature** <item, item-tag>; **UI** ["item", "tag"]

Warnings: "decision conflict" = (permit MT1 A IT) and (deny MT2 A IT) and
(mtags M MT1) and (mtags M MT2)
"declined-friend view "= (permit MT view photo) and (mtags M MT)
and (mtags M declined-friend-request)

Task Actions: *create* on {mtags, itags} ; *delete* on {mtags, itags}

Figure 2: Specifications for a social network.

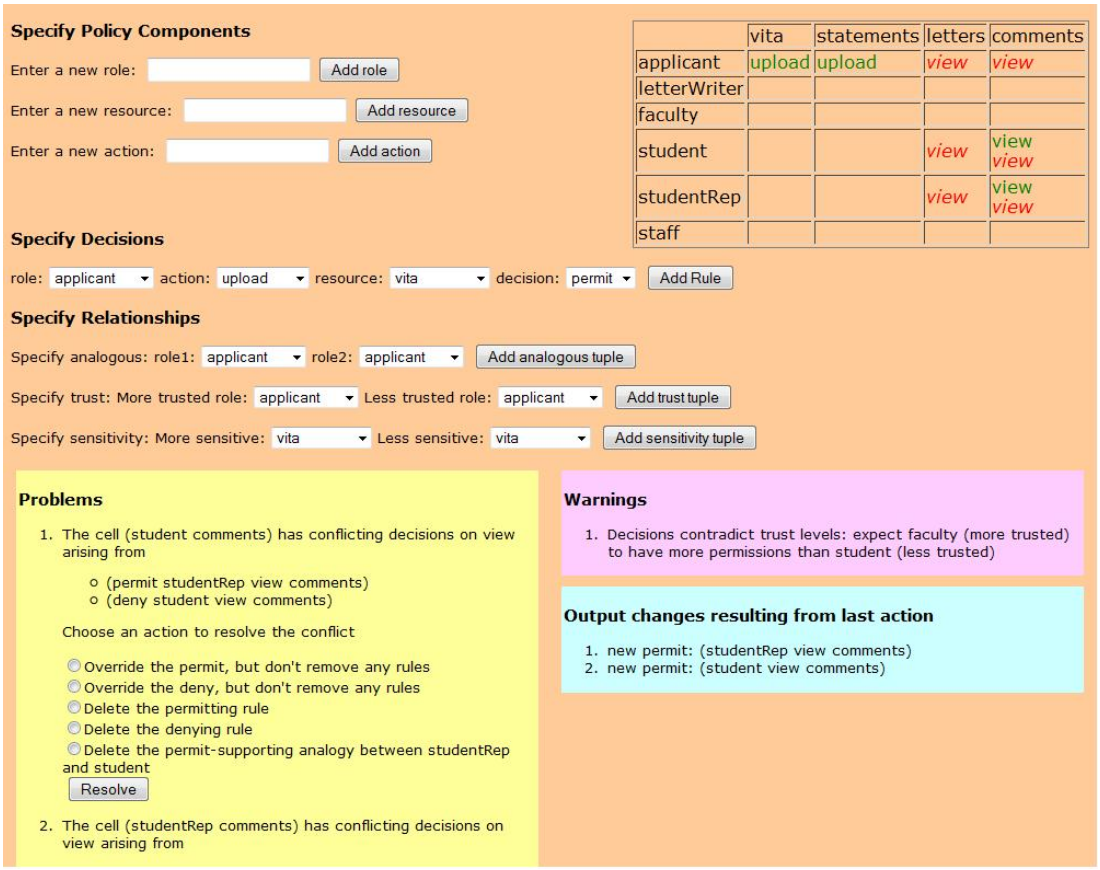

Figure 3: Screenshot of the generated authoring tool for hiring policies. The author has already added the roles, actions, and resources shown in the matrix. The matrix shows both permit and deny decisions (the denies appear in italics, as well as in red if viewed in color). The author has specified that "student" and "studentRep" are analogous, and that "faculty" are more trusted than "student". The content of the "Problems" and "Warnings" areas derive from this information according to the queries in figure 1. The last action performed was entering the decision that "studentRep"s should be allowed to view comments; this induces the contents of the "Output changes ..." area.

output with another for a specific input, QnA stores the input tuple, the identities of the rules leading to the conflicting outputs, and an indication of which rule should take precedence. (An alternative is to use policy combinators, but these generally operate on entire modules, and are a poor fit with incremental policy authoring.) The generated Datalog programs include an overrides relation of the form

overrides(ruleid-use, ruleid-suppress, tuple-component, ...)

that is taken into account by all queries over the output relations. QnA automatically augments designer-defined queries to consult the overrides relation. We also add a unique identifier to each fact that asserts an output decision for a concrete tuple. For example, if a user explicitly states that authors may submit papers, our query evaluator uses a fact like

permit(author,submit,paper,rule67)

to associate the name rule67 with this output decision. This enables us to simulate the notion of proof justification [26], which is built into some systems but not the one we use.

Reporting Changes.

QnA implements change impact between successive pairs of instances. It does so by renaming the old versions of all the relations: e.g., relation RL becomes RLold. Datalog is then given both old and new definitions and asked to find all the tuples that fall in the difference (i.e., in one relation but not the other). These changes are presented to the user, as shown in the bottom right area of figure 3. The changes presented in the screenshot arose from the author specifying a permission on studentRep which propagated to student based on an analogy.

Tool Evaluation

We experimented with QnA on examples similar to and including those in section 3.2. Running queries and computing differences on each iteration exposed no performance issues: the analyses consumed only milliseconds and created no perceivable lag time over the usual web-server response time.

Our experiments suggest future refinements for the model and tool. Consider multiple authoring tools operating over shared data (such as social-network users editing tuples that reside in a central database). The members' specification in the social network defines domains member and item in order to define the mtags and itags relations. Not only should members be prevented from editing member and item, these domains should be inherited from the global system. Furthermore, the contents of the item relation visible to a member should be restricted to items owned by that member.

The observation about shared data raises a system-wide integrity constraint between the owner and itags relations. The social network raised another: the friend-of-friend tag should be computed from the friend tag, not defined by members (as the generated tool currently allows). Our model does not currently support defining one relation in terms of another. Capturing dependencies and integrity constraints between relations might in turn enable warnings about interactions between policy edits and system-wide behavior.

In addition, the warning about conflicting outputs in the social network example suggests that some warnings should allow overriding options. As prompting for overrides on all warnings still seems excessive (as discussed in section 4), this suggests another point of configuration in specifications. Designing a useful form of such a specification remains a problem for future work.

5. RELATED WORK

Policy languages with varying logical foundations abound in the literature. Our model adapts to any policy logic with a useful set of decidable queries. The QnA prototype supports any logic that can be encoded in non-recursive Datalog; this is strictly a subset of the logics that fit our overall model.

Typical policy-authoring tools have tended to focus on how best to elicit and visualize the data in access-control matrices. Zurko, et al.'s ADAGE authoring framework [32] was explicitly designed for usability and to study expressiveness questions such as the use of various grouping mechanisms (such as roles and labels) for expressing policies, but focuses more on interface issues than on the underlying language (with the exception of strong support for separation-of-duty). Brostoff, et al. [4] developed a tool to help scientist end-users author access-policies; their work focuses more on how to help end-users specify policy components. Both used fairly standard rule-based structures for capturing policies, and neither prompts the user as in our sense. Reeder, et al. [23] propose rich visualizations of policy matrices. Karat, et al.'s SPARCLE authoring tool [3] uses natural-language processing to improve the interface between humans and RBAC-style languages. However, focusing on interfaces to RBAC misses the bigger problem—well understood in requirements gathering—of stakeholders not articulating what they actually mean. None of these papers report on cognitive studies justifying the linguistic structures through which people describe policies. None of them elicit or exploit the triangulating information at the heart of QnA.

Reeder, et al. identify five usability challenges for policy authoring tools [24], based on a study in which novice policy authors wrote formal versions of organizational privacy policies. QnA directly addresses three of these: our problem queries detect conflicting decisions; our focus on relationships supports treating policy objects collectively; and our logic allows users to state deny decisions explicitly (rather than only permits). The other two issues were artifacts of the study's use of natural language in policy authoring.

The current QnA system produces authoring tools aimed at access-control requirements gathering. Existing works on requirements engineering for security [1, 12, 29] propose processes to get stakeholders to articulate concerns such as threats, vulnerabilities, trust assumptions, and security goals, as a key step toward developing security requirements. These processes often start from whatever documentation an organization has available about the system to be secured and the staff who will use it. Even if an organization has such documentation (less common in smaller organizations), our experience shows that these sometimes contradict needed privileges. QnA's triangulating checks provide a new, more flexible model for tools to use non-security specific information during authoring.

Building on the work by Palen and Dourish [22], Nissenbaum [19], and Dourish and Anderson [6] (as reviewed in section 2), it may be possible to identify common patterns of triangulating relations and warning queries. Even if these works do suggest general models for triangulation, we expect user-defined auxiliary relations will still be important. Stenning and Van Lambalgen's work [27] highlights that people often reason under (reasonable) assumptions that contradict those of the logical system in which they are working. This body of work, which partly inspired our research, has serious implications for policy authoring: software systems interpret

policy rules under a fixed logic that may differ from what the policy author had in mind. Expanding authoring environments with auxiliary information may help detect such cases, thus leading to more accurate policies.

There are many tools for analyzing and verifying access-control policies. The authors' own Margrave tool [9] performs not only verification but also a sophisticated notion of semantic differencing between policies (the cited paper discusses related tools extensively). Neither Margrave nor the other related tools perform their computation interactively, nor do they have any notions of warnings or triangulation.

There has been considerable attention to handling inconsistency in requirements and software. For instance, Nuseibeh, Easterbrook, and Russo [20] present a manifesto arguing for tolerating and supporting inconsistencies. We believe QnA is consistent with their view, allowing some inconsistencies to be specified as warnings rather than problems (or not be flagged at all), and helping policy authors incrementally resolve these inconsistencies.

Environments that attempt to assist programmers through interactive feedback share common goals with our work. The most famous early example is Teitelman's DWIM for Interlisp [28]. The Programmer's Apprentice [30] used planning to follow and assist programmers. As the programming context is quite different in details from policy authoring, the similarities between these efforts are primarily superficial. There have also been some tools, such as FindBugs [13], that attempt to find errors based on "smell tests". QnA differs from such tools by using triangulation and focusing on asking questions of the policy developer. Other tools use interactive feedback to attempt to refine system specifications automatically. Garcez et al. propose an iterative model for developing requirements specifications [10]. Each iteration of their model checks the partial specification against a set of formal properties and provides diagnostic information explaining failures. The diagnostic information supports learning specifications from properties, a very different task than ours. QnA also differs in its focus on triangulation as a fundamental, human-centric, component of authoring.

6. PERSPECTIVE AND FUTURE WORK

This paper focuses on a model and prototype for triangulating policy-authoring environments. The project also has substantial logical and human factors components, which are beyond the scope of this paper. In addition, we have yet to exploit other interesting observations about authoring practice. This section briefly discusses these components.

6.1 HCI Considerations

The features of our environment that support triangulation and iterative feedback yield a more powerful tool, but the added complexity may make authoring more difficult from a human-factors perspective. This overall project needs to investigate questions such as: how many triangulating relations can one person track through an authoring task?; for what kinds of problems do authors find the triangulating checks helpful?; when should the environment prompt for corrections so as not to disrupt productive authoring sessions? Each of these foundational questions requires carefully designed and controlled user studies. The first round of studies are in progress.

6.2 Logical Considerations

Although policy logics have been explored extensively, maintaining readable policies across incremental additions of exceptions and overrides remains an interesting and open problem. Proper treatment of rules based on relationships such as analogy has not been studied extensively in the policy-logic literature. Our model allows authors to iteratively specify new and complex rules (such as our analogy example). We may need to put constraints on these rules to prevent authors from creating inconsistent policy logics. Understanding these issues better could lead to changes in QnA's rule specifications, but the spirit of these specifications should remain the same.

6.3 Further Forms of Authoring Support

Our experience collecting access-control requirements for faculty hiring (described in section 2) suggests other policy authoring needs that we describe elsewhere [8] but did not build into the QnA prototype.

- **Support Social Concerns as Assets:** People often weigh access decisions against broader issues such as educational value to students and impact on the department's reputation. While these also give alternative views on the policy space, they do not reduce to relations as we used for triangulation. Figuring out how to help authors weigh these issues is an interesting question. For example, the environment might ask authors to identify their social assets, ask for examples of output decisions that would support or threaten those assets, then include checks that the policy is consistent with those decisions.

- **Help Authors Consider Effects of Space and Time:** Access privileges can change over time even as the policy remains fixed: an individual's role might change, or access might depend on the status of a resource (such as whether an application is complete). Role overlaps and changes, in particular, can result in information leaks. We find that authors frequently overlook these issues, which suggests that prompting users about potential changes of these sorts would be a useful aid in authoring.

- **Customize to Authoring Style:** People approach policy authoring from many high-level perspectives: some worry most about potential problems in the event of leakage, some worry more about social impacts of decisions, some prioritize minimal interference with workflow. Each of these styles would probably use an inquisitive environment differently. A framework that could detect and adapt to authoring styles could help a broad range of users. Other works have noted different personality styles in authoring [5, 18, 21], though none offer suggestions on building flexible policy tools.

Acknowledgments.

We thank Keith Stenning for inspiring our interest in how users reason about access-control, Rob Reeder for valuable discussions on the state of authoring tools, and the participants in our hiring-software case study. This work was partially supported by the NSF and a Google Research Award.

7. REFERENCES

[1] A. I. Antón and J. B. Earp. Strategies for developing policies and requirements for secure e-commerce systems. In A. K. Ghosh, editor, *Recent Advances in E-Commerce Security and Privacy*, pages 29–46. Kluwer Academic Publishers, 2001.

[2] P. Bonatti, S. Vimercati, and P. Samarati. A modular approach to composing access control policies. In *Proceedings of the ACM Conference on Computer and Communication Security*, 2000.

[3] C. A. Brodie, C.-M. N. Karat, and J. Karat. An empirical study of natrual language parsing of privacy rules using the SPARCLE policy workbench. In *Symposium on Usable Privacy and Security*, 2006.

[4] S. Brostoff, M. Sasse, D. Chadwick, J. Cunningham, U. Mbanaso, and S. Otenko. R-what? development of a role-based access control (RBAC) policy-writing tool for e-scientists. *Software: Practice and Experience*, 35(9):835–856, 2005.

[5] L. Cranor, J. Reagle, and M. Ackerman. Beyond concern: Understanding net users attitudes about online privacy. Technical Report TR 99.4.3, AT&T Research Labs, Apr. 1999.

[6] P. Dourish and K. Anderson. Privacy, security... and risk and danger and secrecy and trust and identity and morality and power: Understanding collective information practices. Technical Report UCI-ISR-05-1, UCI Institute for Software Research, Irvine, Ca., 2005.

[7] R. B. Findler, J. Clements, C. Flanagan, M. Flatt, S. Krishnamurthi, P. Steckler, and M. Felleisen. DrScheme: A programming environment for Scheme. *Journal of Functional Programming*, 12(2):159–182, 2002.

[8] K. Fisler and S. Krishnamurthi. Escape from the matrix: Lessons from a case-study in access-control requirements. Technical Report CS-09-05, Brown University, 2009.

[9] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies. In *International Conference on Software Engineering*, pages 196–205, May 2005.

[10] A. D. Garcez, A. Russo, B. Nuseibeh, and J. Kramer. Combining abductive reasoning and inductive learning to evolve requirements specifications. *IEE Proceedings-Software*, 150(1):25–38, feb 2003.

[11] S. Godik and T. M. (editors). eXtensible Access Control Markup Language, version 1.1, Aug. 2003.

[12] C. Haley, R. Laney, J. Moffett, and B. Nuseibeh. Security requirements engineering: A framework for representation and analysis. *IEEE Transactions on Software Engineering*, 34(1):133–153, 2008.

[13] D. Hovemeyer and W. W. Pugh. Finding bugs is easy. In *OOPSLA Companion*, pages 132–136, 2004.

[14] S. Jajodia, P. Samarati, and V. S. Subrahmanian. A logical language for expressing authorizations. In *IEEE Symposium on Security and Privacy*, pages 31–42, 1997.

[15] S. Krishnamurthi, P. W. Hopkins, J. McCarthy, P. T. Graunke, G. Pettyjohn, and M. Felleisen. Implementation and use of the PLT Scheme Web server. *Higher-Order and Symbolic Computation*, 20(4):431–460, 2007.

[16] S. Lederer, J. I. Hong, A. K. Dey, and J. A. Landy. Personal privacy through understanding and action: five pitfalls for designers. *Personal and Ubiquitous Computing*, 8(6):440–454, 2004.

[17] E. C. Lupu and M. Sloman. Conflict in policy-based distributed systems management. *IEEE Transaction on Software Engineering*, 25(6):852–869, 1999.

[18] A. D. Miller and W. K. Edwards. Give and take: a study of consumer photo-sharing culture and practice. In *ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 347–356, 2007.

[19] H. Nissenbaum. Will security enhance trust online, or supplant it? In R. M. Kramer and K. S. Cook, editors, *Trust and Distrust in Organizations: Dilemmas and Approaches*, chapter 7, pages 155–188. Russell Sage Foundation, 2004.

[20] B. A. Nuseibeh, S. M. Easterbrook, and A. Russo. Making inconsistency respectable in software development. *Journal of Systems and Software*, 58(2):171–180, 2001.

[21] J. S. Olson, J. Grudin, and E. Horvitz. A study of preferences for sharing and privacy. In *ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 1985–1988, 2005.

[22] L. Palen and P. Dourish. Unpacking "Privacy" for a networked world. In *ACM SIGCHI Conference on Human Factors in Computing Systems*, 2003.

[23] R. Reeder, L. Bauer, L. Cranor, M. Reiter, K. Bacon, K. How, and H. Strong. Expandable grids for visualizing and authoring computer security policies. In *ACM SIGCHI Conference on Human Factors in Computing Systems*, 2008.

[24] R. W. Reeder, C.-M. Karat, J. Karat, and C. Brodie. Usability challenges in security and privacy policy-authoring interfaces. In *INTERACT (2)*, pages 141–155, 2007.

[25] R. Reiter. On closed world data bases. In *Logic and Data Bases*, pages 55–76, 1978.

[26] A. Roychoudhury, C. R. Ramakrishnan, and I. V. Ramakrishnan. Justifying proofs using memo tables. In *Principles and Practice of Declarative Programming*, 2000.

[27] K. Stenning and M. van Lambalgen. *Human Reasoning and Cognitive Science*. MIT Press, 2008.

[28] W. Teitelman. *Interlisp Reference Manual*. Xerox, 1974.

[29] A. van Lamsweerde. Elaborating security requirements by construction of intentional anti-models. In *International Conference on Software Engineering*, pages 148–157, 2004.

[30] R. C. Waters and C. Rich. *The Programmer's Apprentice*. Addison-Wesley, 1990.

[31] T. Whalen, D. K. Smetters, and E. F. Churchill. User experiences with sharing and access control. In *CHI Extended Abstracts*, pages 1517–1522, 2006.

[32] M. E. Zurko, R. Simon, and T. Sanfilippo. A user-centered, modular authorization service built on an RBAC foundation. In *IEEE Symposium on Security and Privacy*, pages 57–71, 1999.