An Administrative Model for Relationship-Based Access Control

Scott D. Stoller(✉)

Department of Computer Science, Stony Brook University,
Stony Brook, USA
stoller@cs.stonybrook.edu

Abstract. Relationship-based access control (ReBAC) originated in the context of social network systems and recently is being generalized to be suitable for general computing systems. This paper defines a ReBAC model, based on Crampton and Sellwood's RPPM model, designed to be suitable for general computing systems. Our ReBAC model includes a comprehensive administrative model. The administrative model is comprehensive in the sense that it allows and controls changes to all aspects of the ReBAC policy. To the best of our knowledge, it is the first comprehensive administrative model for a ReBAC model suitable for general computing systems. The model is illustrated with parts of a sample access control policy for electronic health records in a healthcare network.

1 Introduction

Gates introduced the term relationship-based access control (ReBAC) to describe access control policies expressed in terms of interpersonal relationships in social network systems (SNSs) [8]. While much of the work on ReBAC retains this focus on SNSs, there is a movement to generalize ReBAC to be suitable for general computing systems, by extending it to consider relationships involving all kinds of entities and by increasing the expressiveness of the policy language. The motivation for a general ReBAC framework is that ABAC is not well suited to express policies that depend on relationships involving entities beyond the subject (user) and target (resource) of the access request, especially when those additional entities are identified by following chains of relationships and attribute dereferences.

This paper defines a ReBAC model based on Crampton and Sellwood's RPPM model for ReBAC [3,4]. RPPM is designed to be suitable for general computing systems. It combines ideas from the UNIX access control model, ReBAC, and role-based access control (RBAC).

This material is based upon work supported in part by NSF under Grants CNS-1421893, CNS-0831298, CCF-1248184, and CCF-1414078. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of NSF.

P. Samarati (Ed.): DBSec 2015, LNCS 9149, pp. 53–68, 2015.
DOI: 10.1007/978-3-319-20810-7_4

The most distinguishing feature of our ReBAC model is that it includes a comprehensive administrative model. The administrative model is *comprehensive* in the sense that it allows and controls changes to all aspects of the ReBAC policy. To the best of our knowledge, it is the first comprehensive administrative model for a ReBAC model suitable for general computing systems. Although the details of our administrative model are specific to our RPPM-based ReBAC model, a similar approach can be used to develop comprehensive administrative models for other ReBAC models. For example, a similar approach can be used to develop a comprehensive administrative model for ReBAC models that use modal logic [5,7], instead of path conditions [4], to characterize composite relationships between entities in the system graph.

A central principle underlying the design of our administrative framework is *economy of mechanism*. Saltzer and Schroeder advocate this principle in their classic paper on design of secure information systems [14]. Li and Mao followed this principle in their design of UARBAC, an administrative framework for RBAC. They note that, in that context, this principle means "Use RBAC to administer RBAC" [12]. In our context, it means "Use ReBAC to administer ReBAC". However, RPPM is not sufficiently powerful to support self-administration. We propose a more expressive version of RPPM, which we call "RPPM Modified", abbreviated RPPM^2, that supports self-administration. RPPM^2 also includes some extensions, such as parameters for relationship labels, not required for self-administration but required to express the complex policies in some application domains, including our running example of electronic health records in a healthcare network.

The most challenging aspect of the design of the administrative model is authorization of actions that add or remove authorization rules. The challenge arises from the tension between giving administrators the flexibility to introduce a variety of customized authorization rules and limiting administrators to ensure that they cannot introduce rules that violate desired safety or availability properties. Our approach is to define a strictness order on authorization rules and specify in the policy the least strict rules that each subject is allowed to add or remove.

2 Background: RPPM Model

The RPPM model is defined as follows [4]. The name "RPPM" reflects that the model is based on relationships, paths, and principal-matching.

System Model. A *system model* comprises a set of types T (also called *entity types*), a set of relationship labels R, a set of symmetric relationship labels $S \subseteq R$ and a permissible relationship graph $G_{PR} = (V_{PR}, E_{PR})$, where $V_{PR} = T$ and $E_{PR} \subseteq T \times T \times R$.

System Instance. Given a system model (T, R, S, G_{PR}), a *system instance* is defined by a system graph $G = (V, E)$ and a type function $\tau : V \to T$, where V is the set of entities and $E \subseteq V \times V \times R$. We say G is *well-formed* if for each entity v in V, $\tau(v) \in T$, and for every edge $(v, v', r) \in E$, $(\tau(v), \tau(v'), r) \in E_{PR}$.

Path Condition. Path conditions are defined recursively: $\diamond$ is a path condition; r is a path condition, for all $r \in R$; if π and π' are path conditions, then $\pi_1;\pi_2$, π^+, and $\overline{\pi}$ are path conditions. Informally, $\diamond$ represents the empty path; r represents a path containing exactly one edge, labeled with r; $\pi_1;\pi_2$ represents the concatenation of paths; π^+ represents one or more occurrences, in sequence, of π; and $\overline{\pi}$ represents π reversed. A path condition of the form r or $\overline{r}$ is called an *edge condition*. Every path condition can be rewritten as a *simple path condition*, in which the reversal operator appears only in edge conditions. We assume hereafter that all path conditions are simple.

Request and Principal-Matching Policy. A *request* has the form (s, o, a), where s (the "subject") is an entity requesting to perform action a on entity o (the "object", *i.e.*, target).

A *principal-matching rule* has the form (π, p), where p is an authorization principal and π is either a path condition defined on R or the special symbol $\top$. A principal-matching rule (π, p) matches a request (s, o, a) if (1) π is $\top$, or (2) π is a path condition, and the system graph contains a path from subject s to object o that satisfies π.

A *principal-matching policy* is a list of principal-matching rules. It may contain at most one rule containing $\top$, and that rule, if present, must appear last. A *principal-matching strategy* defines how the principals in matched rules should be combined to obtain the set of principals that match a request. For example, the AllMatch strategy includes in the set the principals from all matched rules in the policy.

Authorization Rule and Authorization Policy. An *authorization rule* has the form $(p, \star, a, d)$ or $(p, o, a,\ d)$, where p is a principal, o is an entity (the "object", *i.e.*, target), a is an action, and $d \in \{0, 1\}$ is the decision ($0 =$ deny, $1 =$ permit). We call "$\star$" the *wildcard*. The rule $(p, \star, a, d)$ says that decision d is a possible decision when principal p requests to perform action a on any object. The rule (p, o, a, d) says that decision d is a possible decision when principal p requests to perform action a on object o. A *permit rule* is an authorization rule whose decision is 1 (permit). A *deny rule* is an authorization rule whose decision is 0 (deny).

An *authorization policy* is a list of authorization rules. A *conflict resolution strategy* determines the decision when rules with different possible decisions match a request; as in XACML [16], the strategies include DenyOverride, FirstMatch, and PermitOverride.

Defaults. The policy must specify a system-wide default decision, which is used when no rules and no other defaults apply. The policy may optionally specify a default decision for each subject and a default decision for each object.

Authorization Algorithm. The algorithm for determining whether a request is authorized has two main stages. First, the request (s, o, a) is matched against the rules in the principal-matching policy, and the principal-matching strategy is applied to obtain a set of matching principals. Second, if the set of matching

principals is non-empty, a list of possible decisions is obtained from the authorization rules with principal p, action a, and object o or $\star$. If the list contains only 0s or only 1s, then that value is the decision. If it contains 0s and 1s, the conflict resolution strategy is applied to obtain a decision. If the list is empty, the default decision for the object is used, if it was specified, otherwise the system-wide default decision is used. If the set of matching principals is empty, then the default decision for the subject is used, if it was specified, otherwise the default decision for the object is used, if it was specified, otherwise the system-wide default decision is used.

3 RPPM^2 Model

This section presents our variant of the RPPM model, which we refer to as "RPPM Modified", abbreviated RPPM^2. Our administrative model, presented in the next section, is for RPPM^2.

System Model and System Instance. The definitions of system model and system instance are unchanged, except that system models are extended to include a set T_{ne} of non-entity types, and relationship labels may have typed parameters. These extensions are not required for our administrative framework. They are introduced to provide the expressiveness needed for our sample policy for a healthcare network, described in Sect. 4. For example, a patient might consent to treatment by a clinician at a particular healthcare facility (possibly also limited to a particular time period or medical episode). This would be reflected in the system model by including the facility as a parameter of the relationship label, *e.g.*, an edge from type Patient to type Clinician labeled with consentToTreatmentAt(Facility), or, for brevity, consent(Facility), where Facility is the type of the parameter.

Request. In RPPM, every request involves exactly two entities, the subject and a single target object. This is insufficient for policy administration, because many administrative actions involve multiple target entities. For example, adding an edge to the system graph involves two equally important target entities, namely, the source and destination nodes of the edge. Therefore, we generalize the definition of actions to allow actions to have any number of parameters, including entities and non-entities, and we do not distinguish any parameter as *the* target. In RPPM^2, a request has the form $(s, op(v_1, ..., v_n))$ where s is the subject (*i.e.*, the entity attempting the operation), op is the operation, and the v_i are arguments. Each operation has a type signature, specifying the type of each parameter.

Path Condition and Path Expression. A *path expression* is defined in the same way as a path condition in RPPM, except, for convenience, we also allow π^*. We change the terminology from "path condition" to "path expression" because, in our framework, a path expression cannot be evaluated to a truth value until the source and target nodes are specified. Each argument of an edge label in a path expression is a wildcard, a constant of the appropriate type, or,

if the parameter has an entity type, a variable. A *path condition* in RPPM^2 has the form $e_1 \cdot \pi \cdot e_2$, where each e_i is an entity constant or a variable, and π is a path condition. A path condition $e_1 \cdot \pi \cdot e_2$ holds if there exists a substitution θ mapping the variables (if any) in the path condition to values such that the system graph contains a path from $e_1\theta$ to $e_2\theta$ that matches $\pi\theta$, where $t\theta$ denotes the result of applying substitution θ to term t.

Principal-Matching Policy. In RPPM, a principal name serves as a shorthand for a path expression expressing a relationship between subject and target. This eliminates the need to repeat that path expression, if it is used in multiple rules. In RPPM^2, because there is not a single distinguished target entity, and because we use path expressions to express relationships between the subject and target entities and between target entities, we replace the notion of principal-matching policy with a simpler and more flexible *path-expression naming* mechanism that simply allows names to be given to path expressions. These names can be used in authorization rules (described below) wherever a path expression is called for. Furthermore, a name can be defined to represent multiple path expressions; a rule in which that name is used expands to multiple rules, one for each path expression associated with the name.

For simplicity, we do not consider path-expression names to be a collectively managed part of the ReBAC policy. Instead, each entity authorized to add authorization rules has its own library of definitions of path-expression names; some might be copied from trusted (e.g., centralized organization-approved) sources, and some might be developed independently. When an authorization rule is added to the policy, the current definitions of path-expression names used in it are stored together with the rule. This allows the rule to be displayed as written (for readability) or in expanded form. The expanded form is used when authorizing the action that adds the authorization rule and when enforcing the authorization rule. If desired, path-expression name definitions can be made a shared and collectively managed part of the ReBAC policy, by defining administrative operations for them and introducing authorization rules for those operations.

Authorization Rule and Authorization Policy. An *authorization rule* has the form $(s, op(a_1, ..., a_n), c, d)$ where:

- the *subject* s is a variable or an entity constant;
- the *action* is an operation op with arguments a_i; each argument with entity type is an entity constant, the wildcard, or a variable; each argument with non-entity type is a constant or a wildcard;
- the *condition* c is a conjunction of path conditions; and
- the *decision* d is 0 (deny) or 1 (permit).

This rule says that decision d is a possible decision when principal p requests to perform action $op(a_1, ..., a_n)$ and the conditions in c are satisfied. Variables that appear in the condition but not in the subject or action are, implicitly, existentially quantified.

As in RPPM, an *authorization policy* is a list of authorization rules, and a *conflict resolution strategy* determines the decision when rules with different possible decisions match a request.

Notation. We adopt several notational conventions for readability and brevity. We write authorization rules as itemized lists of components instead of tuples. We elide the decision when it is 1 (permit). We allow disjunction as a top-level connective in conditions; this is shorthand for a set of rules, one for each disjunct. We use identifiers that start with upper case for types and variables, and identifiers that start with lower case for relationship labels and constants.

4 Example: Healthcare Network

We illustrate our framework with a sample policy for electronic health records at a healthcare network. It is based partly on the sample policy in [9,10]. The permissible relationship graph appears in Fig. 1. The figure also shows the entity types and the relationship labels with their parameter types. Some edges have multiple relationship labels; this is equivalent to putting those relationship labels on separate edges with the same source and target. The figure shows all of the relationship labels in our case study, even though some relationship labels are not used in the sample authorization rules presented in this paper. "humResMgr" abbreviates "human resources manager". "policyOfcr" abbreviates "policy officer".

The healthcare network consists of multiple facilities, such as hospitals and substance-abuse facilities. We assume that the healthcare network maintains a centralized database of health records; separate health records at each facility could easily be modeled. An health record is composed of hierarchically structured items. The "consent" relation indicates that a patient has given consent to treatment by a clinician or workgroup (abbreviated "wkgroup"). A patient's consent to treatment by a clinician may be limited to specified healthcare facilities in the network, so the consent relationship between them is parameterized by facility. Such parameterization is not needed for the consent relation between

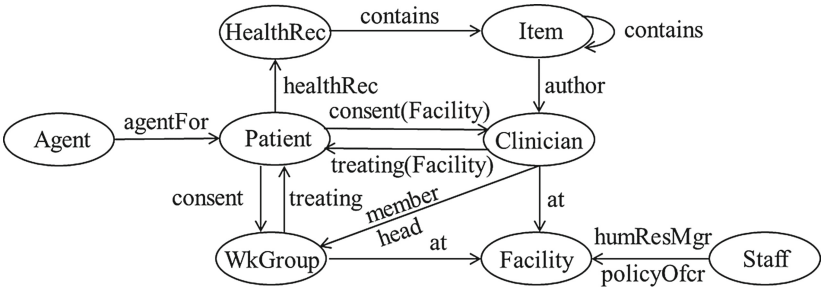

Fig. 1. Permissible relationship graph for the healthcare network example.

patients and workgroups, because a workgroup exists at a single facility. Similarly, the "treating" relationship between clinicians and patients is parameterized by the facility, while the "treating" relationship between workgroups and patients does not need this parameterization. We assume that a patient's agent may represent the patient at any facility in the healthcare network, so the agent-For relation is not parameterized by facility, but this parameter could easily be added if desired.

The following sample authorization rule expresses that a clinician can read an item in a patient's health record if the clinician is treating the patient, is a member of a workgroup treating the patient, or is the item's author.

Subject: Clinician
Action: read(Clinician, Item)
Condition: Clinician · treating($\star$); healthRec; contains$^+$ · Item
$\vee$ Clinician · $\underline{\text{member}}$; treating($\star$); healthRec; contains$^+$ · Item
$\vee$ Clinician · $\overline{\text{author}}$ · Item

The common sub-expression in the first two path conditions could be factored out with a path-expression name definition, *e.g.*, itemOfTreatedPatient = treating($\star$); healthRec; contains$^+$.

For an example of a rule that can be expressed more conveniently in RPPM^2 than ABAC, consider an extension of the system model that includes types of organizational units in the healthcare network, a type for documents such as budgets, a relationship label "orgHrchy" for edges in the organizational hierarchy, a relationship label "manages" to express that a user manages an organizational unit, and a relationship label "budget" to express that a document is an organizational unit's budget. For example, the system graph might contain an entity representing the entire healthcare network with child entities representing facilities with child entities representing centers (e.g., cancer center, heart center) with child entities representing departments (e.g., radiation oncology, surgical oncology) with child entities representing wards. The following rule concisely expresses that the manager of an organizational unit can read the budgets of that organizational unit and its descendants in the organizational hierarchy. This policy cannot be expressed so concisely in traditional role-based access control (RBAC) or attribute-based access control (ABAC) frameworks.

Subject: Manager
Action: read(Manager, Budget)
Condition: Manager · manages; orgHrchy*; budget · Budget

5 Authorization Checking Algorithm

We sketch an authorization checking algorithm for RPPM^2, based on Crampton and Sellwood's authorization checking algorithm for RPPM [4]. The heart of

their authorization checking algorithm is an algorithm, called MatchPrincipal, that, given a system graph, a path expression, a source node, and a target node, performs a modified bread-first search (BFS) to determine whether the graph contains a path from the source node to the target node that matches the path expression.

Two main extensions to their authorization checking algorithm are needed to obtain an algorithm for determining whether a RPPM^2 authorization rule applies to a given request. First, MatchPrincipal needs to be extended to match parameters in relationship labels as it traverses edges. This requires maintaining a substitution that maps the existentially-quantified variables to values. The algorithm adds and removes bindings from this substitution as it traverses edges and backtracks across them, respectively.

Second, the overall authorization checking algorithm needs to be extended to check all of the path conditions in the condition c of a given authorization rule. Let pc_i denote the path condition in the i'th conjunct of c. For each i, starting with $i = 1$, a BFS for pc_i is initiated, using the substitution obtained from the BFS for pc_{i-1} as the initial substitution (as a special case, when $i = 1$, the empty substitution is used as the initial substitution). When a path and associated substitution satisfying pc_i is found, a BFS for pc_{i+1} is initiated, in the same manner. If the BFSs for all path conditions succeed, the overall algorithm terminates with the result that the authorization rule applies to the given request. If the BFS for pc_i fails, then the BFS for pc_{i-1} resumes until it finds a satisfying path with a different associated substitution, at which point a new BFS for pc_i is initiated. This process iterates until either the BFSs for all path conditions succeed, yielding the result mentioned above, or the BFS for some path condition pc_i terminates, in which case there are no more ways to try to satisfy pc_i, and the overall algorithm terminates with the result that the authorization rule does not apply to the given request.

One issue glossed over above is that, when a BFS for some pc_i is initiated, the source node for the BFS could be an unbound variable. In principle, this can easily be handled by adding a loop that considers every node in the system graph as a potential source node. In practice, this would be unacceptably inefficient, and the authorization rule should probably be rejected if this iteration cannot be avoided; this can be determined with a simple static analysis, so the authorization rule can be rejected before it is added to the policy. We expect that this will rarely occur in practice. Even if the source node in some path condition pc_i is an existentially quantified variable V, two techniques can be used to try to avoid this iteration: (1) if the target node for pc_i is known (*i.e.*, is not an unbound variable), reverse the path expression, and swap the source and target nodes; (2) change the order in which the path conditions in c are considered (in effect, permute the conjuncts of c), so that some path condition that has a known source or target node and that contains V (hence will generate a binding for V) is considered before pc_i.

6 Administrative Model

There are administrative operations to add and delete entities, edges, and authorization rules, plus three administrative actions to set defaults. We discuss administrative policy for each of them in turn. As discussed in Sect. 1, administrative policies are expressed in the same way as non-administrative policies, using authorization rules, as defined above. Any desired authorization rules may be used to control these administrative operations. However, authorization rules for actions that add or delete rules are given special interpretations, as discussed in detail below. We refer to authorization rules for administrative actions as *administrative rules* or *administrative policy*. Our administrative model is comprehensive: even the administrative policy can be modified dynamically, using the same administrative operations.

We sometimes refer to entities authorized to perform one or more administrative actions as *administrators*. These entities are not special in any other way; for example, they may also have non-administrative permissions.

Add Edge and Delete Edge. The action addEdge(e_1, e_2, r) adds an edge from e_1 to e_2 labeled r. If the edge already exists, the action has no effect. For example, the following authorization rule expresses that a clinician who is a member of a workgroup WG can create a treating relationship between WG and a patient, if the patient has given consent to treatment by WG.

Subject: Clinician
Action: addEdge(WG, Patient, treating)
Condition: Clinician · member · WG ∧ Patient · consent · WG

The action deleteEdge(e_1, e_2, r) deletes an edge from e_1 to e_2 labeled r. If the edge does not exist, the action has no effect. For example, the following authorization rule expresses that a human resources manager for a facility can remove a clinician's association with the facility.

Subject: HumResMgr
Action: deleteEdge(Clinician, Facility, at)
Condition: HumResMgr · humResMgr · Facility

Add Entity and Delete Entity. We require that, when a new entity is added, an edge relating it to an existing entity is also added. This provides a reference point to determine permissions. For example, a typical pattern is to add an "owner" edge between the subject and the new entity. This requirement is analogous to the requirement in ARBAC97 that, when a new role is added, it must be attached to the existing role hierarchy [15]. When such a reference is not needed (*e.g.*, a user is authorized to add new entities of a given type anywhere in the system graph), the new entity can be related to a dummy entity.

The action addEntity(T, e_1, e_2, r) adds a new entity e_1 of type T, and adds an edge labeled r from e_1 to existing entity e_2. If e_1 already exists, the action has no effect. For example, the following authorization rule expresses that a human resources manager for a facility can add new clinicians at the facility.

Subject: HumResMgr
Action: addEntity(Clinician, NewClinician, Facility, at)
Condition: HumResMgr · humResMgr · Facility

The action deleteEntity(e_1) deletes entity e_1 and all edges incident on e_1. This action is permitted only if the subject is authorized to delete e_1 and to delete all edges currently incident on e_1. For example, the following authorization rule expresses that the human resources manager for a facility can remove workgroups at that facility. This action is permitted only if the workgroup is not currently treating any patients, because human resources managers are not authorized to delete treating relationships between workgroups and patients. The Facility variable in this rule does not appear in the subject or action and hence is existentially quantified.

Subject: HumResMgr
Action: deleteEntity(WG)
Condition: HumResMgr · humResMgr · Facility $\wedge$ WG · at · Facility

Add Authorization Rule. The action addRule$(s, op(a_1, ..., a_n), c, d)$ adds the specified authorization rule to the authorization policy.

We could authorize such actions in exactly the same way as other actions, using authorization rules with addRule actions, without any special treatment, but this would be undesirably limiting, because each administrator would only be able to select from a finite, fixed set of rules that he or she is permitted to add. In practice, more flexible administrative policies are usually desired, that allow an administrator to add authorization rules selected from a large or unbounded set of authorization rules that are consistent with the administrator's scope of authority and with desired safety and availability properties. Safety properties require that, under specified conditions, specified entities do not have specified permissions. Availability properties require that, under specified conditions, specified entities have specified permissions. Note that these properties depend on the administrative policy as a whole, not only the policies for adding and removing rules. Support for verifiable enforcement of such properties is an important design principle for our administrative model, although details of specification and verification of such properties are left for future work.

Based on this consideration, one might design an administrative policy for addRule actions that directly specifies the set of authorizations that each entity is permitted to grant by adding permit rules, and the set of authorizations that each entity is permitted to revoke by adding deny rules. An *authorization* is a pair consisting of an entity and an action, representing permission for that entity to perform that action. In this approach, an addRule action for a permit rule

is allowed if the proposed rule grants a subset of the specified authorizations in the current state, *i.e.*, with respect to the current system graph. Similarly, an addRule action for a deny rule is allowed if the proposed deny rule revokes a subset of those authorizations in the current state.

This approach allows a wider variety of rules to be added, but it is brittle and potentially insecure: the authorization check for addRule actions is too dependent on the current state, and added rules might have undesired consequences in other states, *e.g.*, after the system graph changes. This is true regardless of whether the two aforementioned sets of authorizations are specified explicitly by enumeration or implicitly using formulas (*e.g.*, using a combination of first-order predicate logic and path conditions). For example, an administrator could circumvent the intended administrative policy by first adding a permit rule that grants dangerous authorizations only if a certain relationship exists in the system graph, and later adding that relationship.

Our approach is based on using logical implication to compare a proposed authorization rule to specified authorization rules that represent the "loosest" (least strict) authorization rules that the subject is allowed to add. A key feature of this approach is that the authorization test for an addRule action is independent of the current state and hence can ensure that the meaning of a proposed rule is acceptable in all states. We introduce a partial order on authorization rules, called *strictness order*. Informally, rule ρ_1 is at least as strict as rule ρ_2, denoted $\rho_1 \leq_{\text{rule}} \rho_2$, if they have the same action and the same decision, and the condition in ρ_1 entails (*i.e.*, implies) the condition in ρ_2. The strictness order is designed to provide the guarantee: if $\rho_1 \leq_{\text{rule}} \rho_2$, then in every state, ρ_1 grants (or denies) a subset of the authorizations that ρ_2 grants (or denies).

An addRule(ρ_1) action by a subject s is allowed if there exists an authorization rule ρ_2 such that (1) the authorization rules imply s can add ρ_2 and (2) ρ_1 is at least as strict as ρ_2.

Roughly speaking, a rule ρ_1 is at least as strict as a rule ρ_2 if ρ_1 can be obtained from ρ_2 by renaming variables, instantiating variables with constants, adding conjuncts to the condition, and replacing path expressions in the condition with stricter path expressions. A formal definition of the strictness order follows.

A substitution θ is a mapping from variables to variables and constants. Recall that $t\theta$ denotes the result of applying substitution θ to term t.

Authorization rule $\rho_1 = (s_1, a_1, c_1, 1)$ is *at least as strict as* authorization rule $\rho_2 = (s_2, a_2, c_2, 1)$, denoted $\rho_1 \leq_{\text{rule}} \rho_2$, if there exists a substitution θ such that $s_1 = s_2\theta$ and $a_1 = a_2\theta$ and, for each conjunct $e_2 \cdot \pi_2 \cdot e_2'$ in condition $c_2\theta$, there is a conjunct $e_1 \cdot \pi_1 \cdot e_1'$ in condition c_1 such that $e_1 = e_2$, $e_1' = e_2'$, and $\pi_1 \leq_{\text{pe}} \pi_2$.

The strictness order $\leq_{\text{pe}}$ on path expressions is designed to provide the guarantee: if $\pi_1 \leq_{\text{pe}} \pi_2$, then in every system graph, every path satisfying π_1 also satisfies π_2. Informally, a path expression π_1 is at least as strict as a path expression π_2, denoted $\pi_1 \leq_{\text{pe}} \pi_2$, if π_1 can be obtained from π_2 by replacing path expressions involving transitive closure with path expressions denoting paths

with the same or more constrained labels and the same or more constrained lengths (*e.g.*, replacing π^+ with $\pi;\pi$). A formal definition of $\leq_{\text{pe}}$ follows. It is unnecessary (although safe) to allow instantiation of variables with constants in this definition, because the instantiation can be done by the substitution applied to c_2 in the definition of $\leq_{\text{rule}}$.

Path expression π_1 is *at least as strict as* path expression π_2, denoted $\pi_1 \leq_{\text{pe}} \pi_2$, if any of the following conditions hold: (1) $\pi_1 = \pi_2$; (2) π_2 has the form π^+ and π_1 either (a) has the form $\pi_{1,0}$ or $\pi^+_{1,0}$ with $\pi_{1,0} \leq_{\text{pe}} \pi$, or (b) is a sequential composition $\pi_{1,1};\pi_{1,2}$ such that, for $i = 1..2$, $\pi_{1,i}$ has the form $\pi_{1,i,0}$, $\pi^+_{1,i,0}$, or $\pi^*_{1,i,0}$ with $\pi_{1,i,0} \leq_{\text{pe}} \pi$, and such that $\pi_{1,1}$ and $\pi_{1,2}$ do not both have * as the top-level operator; (3) π_2 has the form π^* and either π_1 satisfies the conditions in case (2) or π_1 is $\diamond$; (4) π_2 has the form $\pi^*_{2,1};\pi_{2,2}$ or $\pi_{2,2};\pi^*_{2,1}$ and π_1 satisfies $\pi_1 \leq \pi_{2,2}$; (5) π_2 has the form $\pi_{2,1};\cdots;\pi_{2,n}$ and π_1 has the form $\pi_{1,1};\cdots;\pi_{1,m}$ and there exist i and j such that $1 < i < m$ and $1 < j < n$ and $\pi_{1,1};\cdots;\pi_{1,i} \leq_{\text{pe}} \pi_{2,1};\cdots;\pi_{2,j}$ and $\pi_{1,i+1};\cdots;\pi_{1,m} \leq_{\text{pe}} \pi_{2,j+1};\cdots;\pi_{2,n}$. Case (5) reflects the associativity of sequential composition.

For example, the following authorization rule expresses that the policy officer for a facility can add rules allowing a member of a workgroup to establish a treating relationship with a patient if the patient has consented to that relationship.

Subject: PolicyOfcr
Action: addRule(**Subject:** Clinician
Action: addEdge(WG, Patient, treating)
Condition: Clinician · member · WG
$\wedge$ Patient · consent · WG)
Condition: PolicyOfcr · policyOfcr · Facility

Note that the authorization rule in the addRule action is the example authorization rule for addEdge given above. This authorization rule for addRule allows a policy officer at a facility to add either the authorization rule as given here, meeting the minimal safety requirements imposed by the healthcare network, or a stricter version imposing additional facility-specific requirements, *e.g.*, that only the head of a workgroup can establish a treating relationship between the workgroup and a patient.

Delete Authorization Rule. The action deleteRule$(s, op(a_1, ..., a_n), c, d)$ deletes the specified authorization rule from the authorization policy. Authorization rules for deleteRule actions are interpreted in the same was as authorization rules for addRule actions. This implies that, if the policy contains an authorization rule for addRule and an authorization rule for deleteRule with the same subject, the same condition, and the same authorization rule in the actions, then each user has symmetric permissions, *i.e.*, the set of authorization rules that can be

added by a user equals the set of authorization rules that can be removed by that user.

Set System-Wide Default and Set Conflict-Resolution Strategy. The action setSystemDefaultDecision(d) sets the system-wide default decision to d. The action setConflictResolutionStrategy(s) sets the conflict-resolution strategy to s. The system-wide default decision and the conflict resolution strategy may be fixed in many systems, in which case no authorization rules for these actions are needed. For systems in which, for example, a system security officer can change the system-wide default decision, the system graph might contain an entity named "sso", corresponding to the system security officer role, and the policy might contain the authorization rule

Subject: SSO
Action: setSystemDefaultDecision(Default)
Condition: SSO · member · sso

Set Per-Subject and Per-Object Defaults. setSubjectDefaultDecision(s, d) sets the default decision for subject s to d. setObjectDefaultDecision(o, d) sets the default decision for object o to d. For example, a policy with ownership-based authorization for setting per-object defaults might contain the authorization rule

Subject: Owner
Action: setObjectDefaultDecision(Obj, Default)
Condition: Owner · owner · Obj

If objects are organized in a hierarchy (*e.g.*, in folders) represented by the "contains" relation, and ownership propagates down the hierarchy, then the rule might be

Subject: Owner
Action: setObjectDefaultDecision(Obj, Default)
Condition: Owner · owner ; contains* · Obj

7 Related Work

Related Work on ReBAC. Several ReBAC frameworks have been proposed, by Fong [5], Carminati, Ferrari, and Perego [1], Cheng, Park, and Sandhu [2], Hu, Ahn, and Jorgensen [11], and others. We chose RPPM [3,4] as the basis for our work, because, as Crampton and Sellwood discuss [4], it is designed to be suitable for general computing systems, in contrast to other ReBAC models designed primarily with social network systems in mind.

Our RPPM^2 model extends the generality and expressiveness of RPPM in several ways, in addition to providing a comprehensive administrative model.

RPPM[2] allows relationship labels to have parameters. It allows actions to involve multiple entities and other parameters rather than only one object. It allows authorization rules to depend on relationships between all pairs of entities involved in a request, and to depend on multiple relationships between each such pair of entities. It allows entity constants as the source or target of path conditions; in RPPM, the source and target of the path condition must be the subject and object of the request, respectively.

In [5], Fong introduces the concept of *context* in a ReBAC model for social network systems. Contexts are scopes in which relationships (*i.e.*, edges in the social network graph) exist. Contexts are hierarchically organized, and the creation and destruction of contexts follow a stack discipline. The usefulness of contexts is illustrated in [5] in a sample policy for electronic health records (without considering administration of that policy). For example, a treating relationship between a clinician and a patient can be limited to a context corresponding to a healthcare facility or a medical case. We achieve a similar effect by allowing relationship labels to have parameters. Parameters can be used to indicate the scope of a relationship, among other things. Relationship labels are less structured than the contexts in [5] but more flexible; for example, they do not impose a strictly hierarchical organization on scopes or a stack discipline on the management of the scopes.

Related Work on Administration of ReBAC. Crampton and Sellwood [3,4] do not consider administration of RPPM.

Fong [5] defines transitions that push and pop contexts and a transition that adds and removes edges in the social network, but does not discuss authorization of those transitions, except for the comment that "In principle, the transitions can be considered resources and thus protected by the same protection system". Furthermore, Fong assumes in [5] that access control policies "remain constant throughout system lifetime", and he does not define operations for changing access control policies or consider authorization of such operations.

Fong, Anwar, and Zhao [6] define a ReBAC model for Facebook-style social network systems (SNSs), including administrative operations that allow each user to set policies for access to his/her own profile and items within it, and policies for communication (e.g., messaging). That model has some SNS-specific features not included in our model, *e.g.*, communication policies. However, that model is developed with SNSs in mind and is less applicable to general computing systems than RPPM and our model. Policy administration in that model is much simpler, because each user sets only policy for access to his/her own resources, and because the model does not use path conditions, modal logic, or any similarly expressive formalism to characterize paths in the graph.

Hu, Ahn, and Jorgensen [11] consider multiparty access control (MPAC) in the context of online social networks. MPAC allows multiple users to jointly determine the access control policy for a resource, *e.g.*, a photo of a group of users. The core of their approach is a flexible voting-based conflict-resolution scheme to handle situations in which the resource's owner and other relevant

users, called *stakeholders*, have different access control preferences for a resource. Our model does not support voting-based MPAC but could be extended to do so.

Related Work on Administration of ABAC. Gupta, Stoller, and Xu define an ABAC model with a comprehensive administrative model [9,10]. Specifically, they define a policy language with Datalog-like authorization rules, and an administrative model based on operations that add and remove facts and rules in the policy. They define a strictness order on authorization rules, and they interpret authorization rules for addition or deletion of authorization rules to allow addition or deletion, respectively, of rules that are at least as strict as the specified rules. We adopt this approach in the design of RPPM^2. They also present an abductive policy analysis algorithm, specifically, a semi-decision procedure for determining whether a specified set of users can together change the policy to grant a specified authorization.

8 Conclusion

This paper presents an expressive ReBAC model that includes a comprehensive administrative model. There are many directions for future work. One direction is to develop a similar comprehensive administrative model for a ReBAC model that uses modal logic [5,7], instead of path conditions [4], to characterize composite relationships between entities in the system graph. A second direction is to optimize the authorization checking algorithm sketched in Sect. 5, using techniques such as: pre-computation and caching, which are discussed for RPPM in [3]; more aggressive exploitation of constraints from multiple path conditions in a rule to prune the search for a path satisfying the overall condition; and elimination of redundant computations for policies in which path conditions in the same or different authorization rules contain common subexpressions. Another approach to developing an efficient authorization checking algorithm is to try to adapt Liu and Stoller's approach to efficient implementation of complex graph queries [13]. A third direction for future work is to develop policy analysis algorithms, to help policy developers understand the full implications of proposed administrative policies. We plan to explore an abductive policy analysis for RPPM^2 along the lines of Gupta, Stoller, and Xu's abductive policy analysis for ACAR [9,10].

References

1. Carminati, B., Ferrari, E., Perego, A.: Enforcing access control in Web-based social networks. ACM Trans. Inf. Syst. Secur. **13**(1), 1–38 (2009)
2. Cheng, Y., Park, J., Sandhu, R.: A user-to-user relationship-based access control model for online social networks. In: Cuppens-Boulahia, N., Cuppens, F., Garcia-Alfaro, J. (eds.) DBSec 2012. LNCS, vol. 7371, pp. 8–24. Springer, Heidelberg (2012)
3. Crampton, J., Sellwood, J.: Caching and auditing in the RPPM model. In: Mauw, S., Jensen, C.D. (eds.) STM 2014. LNCS, vol. 8743, pp. 49–64. Springer, Heidelberg (2014)

4. Crampton, J., Sellwood, J.: Path conditions and principal matching: a new approach to access control. In: Proceedings of the 19th ACM Symposium on Access Control Models and Technologies (SACMAT), pp. 187–198. ACM (2014)
5. Fong, P.W.L.: Relationship-based access control: protection model and policy language. In: Proceedings of the First ACM Conference on Data and Application Security and Privacy (CODASPY), pp. 191–202. ACM (2011)
6. Fong, P.W.L., Anwar, M., Zhao, Z.: A privacy preservation model for Facebook-style social network systems. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 303–320. Springer, Heidelberg (2009)
7. Fong, P.W.L., Siahaan, I.: Relationship-based access control policies and their policy languages. In: Proceedings of the 16th ACM Symposium on Access Control Models and Technologies (SACMAT), pp. 51–60. ACM, June 2011
8. Gates, C.E.: Access control requirements for Web 2.0 security and privacy. In: IEEE Web 2.0 Security & Privacy Workshop (W2SP 2007), May 2007
9. Gupta, P.: Verification of security policy administration and enforcement in enterprise systems. Ph.D. thesis, Stony Brook University, December 2011. https://dspace.sunyconnect.suny.edu/handle/1951/59677
10. Gupta, P., Stoller, S.D., Xu, Z.: Abductive analysis of administrative policies in rule-based access control. IEEE Trans. Dependable Secure Comput. **11**(5), 412–424 (2014)
11. Hu, H., Ahn, G.J., Jorgensen, J.: Multiparty access control for online social networks: model and mechanisms. IEEE Trans. Knowl. Data Eng. **25**(7), 1614–1627 (2013)
12. Li, N., Mao, Z.: Administration in role based access control. In: Proceedings of the 2nd ACM Symposium on InformAtion, Computer and Communications Security (ASIACCS), pp. 127–138. ACM, March 2007
13. Liu, Y.A., Stoller, S.D.: Querying complex graphs. In: Van Hentenryck, P. (ed.) PADL 2006. LNCS, vol. 3819, pp. 199–214. Springer, Heidelberg (2005)
14. Saltzer, J.H., Schroeder, M.D.: The protection of information in computer systems. Commun. ACM **17**(7), 388–402 (1974)
15. Sandhu, R., Bhamidipati, V., Munawer, Q.: The ARBAC97 model for role-based administration of roles. ACM Trans. Inf. Syst. Secur. **2**(1), 105–135 (1999)
16. eXtensible Access Control Markup Language (XACML). http://www.oasis-open.org/committees/xacml/