Towards Automated Negotiation of Access Control Policies

Vijay G. Bharadwaj and John S. Baras
Institute for Systems Research, University of Maryland,
College Park MD 20742, USA.
{vgb,baras}@umd.edu

Abstract

We examine the problem of negotiating access control policies between autonomous domains. Our objective is to develop software agents that can automatically negotiate access control policies between autonomous domains with minimal human guidance. In this paper we show a mathematical framework that is capable of expressing many such negotiation problems, and illustrate its application to some practical scenarios.

1 Introduction

As computer systems become more and more interconnected, many situations arise where different systems need to share data or resources. For instance, a provider who wants to offer a number of services over the web must decide how much a remote user with a certain set of credentials is to be trusted. A more complicated example is provided by collaborative computing in peer-to-peer networks, which frequently requires two or more autonomous domains to share data or other resources to achieve a common goal. Often, the collaboration itself may generate new data or resources, and these must also be shared.

In all the above situations, the domains involved must agree upon an access control policy for their shared resources. Currently, this is done by a variety of methods, all of which require human intervention. Thus, many of these methods are time-consuming, inflexible and error-prone. For client-server scenarios such as the web services scenario above, a policy might be set by a human administrator in advance assigning remote users to local roles. For peer-to-peer networks such as military coalitions, negotiations are carried out by human beings meeting in person, through a tedious process of discussion and bargaining that can take weeks or months to conclude.

We examine the problem of automating the negotiation of access control policies between autonomous security domains. Specifically, we examine the problem of negotiating a shared access state, assuming all domains use an RBAC policy model. We propose a mathematical framework in which many such problems can be cast, and show that this framework is expressive enough to deal with a large number of practical situations. Further, our framework provides an efficient means for auditing permissions in an access control system, and for checking access control states against higher-level security policies.

In our target scenario, coalition formation would take place as follows. First, administrators from a number of domains would agree to try and set up a coalition to carry out a mission important to all of them. Each domain would have a software agent to help negotiate policy with other domains. The administrators would program their individual agents with some higher-level considerations, such as the mission objective, the applications that must be run in the coalition to achieve the objective, its importance to that domain, the (real or perceived) trustworthiness of the other partners, and so on. These software agents would then communicate with each other and arrive at a common access control policy for the coalition, which spelled out details such as the accesses permitted to members of particular roles in particular domains, the separation of duty relationships in the system, and the permissions to be applied to new objects created during coalition operation.

The novelty of our work lies in the fact that in our framework, the automated negotiation agent can be guided not only by hard constraints such as domain meta-policies, but also by soft constraints which reflect the preferences of the domain. This allows the agent to optimize; it is now looking for the best access control policy compatible with its constraints, as opposed to looking for any policy that meets its domain's requirements.

In Section 2 we sketch the negotiation problem and provide an outline of our work. In Section 3 we provide some theoretical background to our work. Section 4 discusses the architecture of the negotiating agent. Section 5 describes our mathematical framework, and its application to some example scenarios. Finally, in Section 6 we consider some directions for future research.

Proceedings of the 4th International Workshop on Policies for Distributed Systems and Networks (POLICY'03)
0-7695-1933-4/03 $17.00 © 2003 IEEE

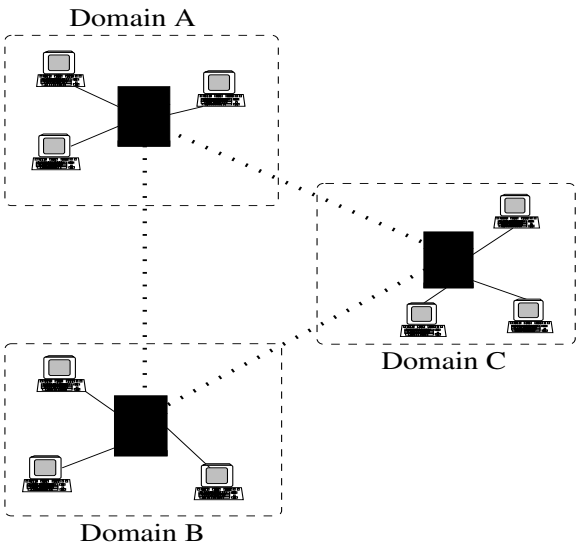

Figure 1. Negotiation in multi-domain networks.

2 The Negotiation Problem

Consider a coalition such as the one in Figure 1. We assume that negotiation is carried out between software agents that are guided by human operators, and that each domain has a single agent that acts as an authorized representative for the domain. These agents must arrive at a set of objects to share and an access control policy for these objects. This policy will then be implemented, perhaps subject to a human operator's approval.

We assume that each domain has its own access control policies, and that the negotiated state must at least satisfy the access control constraints of all the domains. For example, if a domain requires a separation-of-duty relationship between two privileges, then the final state must not have any user with both privileges.

In addition to the domain policy, each domain's administrator will have to provide the domain's negotiating agent with some information regarding the coalition that is to be set up, such as what data or applications are to be shared, or what services need to be available to coalition members. Administrators can also provide information on features they consider desirable in a coalition access control state. However, the existing domain policy already contains a lot of information about the domain's preferences; for instance, objects that are only accessible to top level managers are presumably more valuable than objects that are accessible to any user in the system. By taking advantage of this existing structure, we avoid duplication of administrator effort. At the same time, the administrator has the flexibility to override selected constraints or to impose additional requirements.

In any given situation, it may not be possible to find any state that satisfies the policies of all the domains. Alternatively, there may be a large number of states that do so. We would like our negotiation agents to be able to find the best possible solution in each case. If all policies cannot be satisfied, we would like to find a state that satisfies as much of the policies as possible; if many states exist that satisfy all domain policies, we would like to pick the best among them according to some criterion.

In many practical situations, access control policies are themselves considered sensitive information. For example, the access control policies of a corporation may reveal some information about its internal business processes, or about confidential relationships with other entities. As a result, our negotiation agents will not have complete security policies to work with; instead, they will have to make do with the limited knowledge gained during the negotiating process. Additionally, the agents must also try to avoid revealing any more information than strictly necessary about their respective domain policies.

There are situations in which no policies are sensitive. For example, the coalition might consist of a single large corporation, with different departments or divisions as its member domains. Even in such cases, the negotiation problem is a hard one. If each policy consists of a set of Boolean constraints, then finding a state to satisfy all the policies is an instance of the satisfiability problem, which is known to be NP-complete. Furthermore, this approach does not deal gracefully with overconstrained problems (no possible solution) or with problems where we must find the best of a large number of feasible solutions.

In this paper we look at the case where policies and domain preferences are not considered sensitive. We confine ourselves to Role Based Access Control (RBAC, [17]) systems, though our methods can be generalized to other access control models. We show that the RBAC constraints, which are derived from higher-level security policies, can be expressed as constraints over an appropriate semiring. We show two such formulations and compare their properties.

We do not attempt to develop a new language for specifying access control policies, nor do we require that all domains specify their policies in the same language. Instead, our framework can be thought of as a useful internal representation for the negotiating agents, to which constraints from the domain policy languages can be easily translated. In Section 5, we show that our representation is general enough to subsume at least some existing policy languages.

We demonstrate the application of our method to two types of scenarios: the multi-domain scenario described above, and to the simpler case of a client-server system.

In the client-server model, there are exactly two domains, and all the shared resources belong to one of the domains (namely, the server). Though much simpler, the client-server model is important in practice because it represents, for instance, web-based access to computing resources. It also builds some intuition that is useful when investigating the multi-domain case.

3 Background

3.1 Access Control Policies

The problem of resource sharing among autonomous domains is a relatively new one in Computer Science. In most classical work in resource sharing, all resources are part of a single system, and there is a single mechanism enforcing access control policies. By contrast, in coalition scenarios, there are often multiple independent authorities (one or more per domain) that perform security functions such as authenticating users and enforcing access control policy.

The presence of multiple domain authorities and policy enforcement points raises the question of how any domain can trust the other domains in the coalition to correctly enforce any coalition-wide access control policy. We will not investigate this problem in any detail; we assume that some form of joint administration is used, or there is some other mechanism for monitoring the compliance of domains.

Role Based Access Control (RBAC) is a widely used model for access control systems. RBAC policies are formulated around the concept of a role. A complete RBAC model [17] includes the following state variables:

- The sets U (users), R (roles), P (permissions) and S (sessions).
- $PA : R \rightarrow 2^P$, a mapping from roles to their associated permissions.
- $UA : U \rightarrow 2^R$, a mapping from users to their roles.
- $user : S \rightarrow U$, a mapping from sessions to the respective users.
- $roles : S \rightarrow 2^R$, a mapping from sessions to the set of roles associated with each session.
- A partial ordering $\geq$ on R which defines a role hierarchy. $R_1 \geq R_2$ implies that R_1 inherits all the permissions of R_2.
- A collection of constraints which specify acceptable combinations of values for *PA*, *UA*, *user* and *roles*.

In practice, the constraints form an important part of the state; they can be used to specify high level organizational policies, such as "no employee shall perform both consulting and auditing services for the same client". RCL2000 [1] is a language for specifying such constraints in an RBAC system. The authors show that RCL2000 is sound and complete with respect to a restricted subset of First Order Logic. They also argue that properties such as Separation of Duty are better specified in terms of permissions than in terms of roles.

A number of languages have been proposed for expressing access control policies, and in particular RBAC policies, efficiently and unambiguously. Ponder [7] is an object-oriented and strongly-typed declarative language for RBAC policies. It provides access control and obligation policies, as well as facilities for policy composition. An interesting feature of Ponder is its classification of constraints into two types: basic constraints and meta-policies. Basic constraints apply to individual policies and limit the applicability of the policy based on specific conditions. Meta-policies apply to groups of policies, such as within a composite policy, and limit the permitted policies in the system. For example, static separation of duty can be specified as a meta-policy. All RCL 2000 constraints can be specified as meta-policies. However, a weakness of Ponder is that more complicated policies like dynamic separation of duty cannot be specified without adding attributes to the objects regulated by the policy.

The Tower policy specification language [13] was developed by applying a language-based approach to authorization. Tower provides for set-valued variables and basic operations on these variables. The structures supported are privileges, permissions, roles, users, ownership and blocks. Roles are specified in Tower along with constraints which limit the roles a user may take on, as well as the roles that a user may have active in the same or concurrent sessions. The role definition also specifies the permissions associated with a role. For users, a set of active role memberships and sessions is maintained at all times. These properties allow Tower to easily express even complicated constraints such as dynamic separation of duty within role definitions.

Another interesting feature of Tower is its support for joint ownership and administration of objects. Each object may be owned by a user, a set of users, a role, a set of roles, or some mix of the above. Tower can also specify how many of these owners must agree if any operation requiring owner approval is to be performed on the object, as well as the ownership of any future objects created through use of the object. For example, one can specify policies that decide the ownership of future documents created by a text editor, depending on which user's access to the editor created the document.

Bonatti et al. [6] propose an algebra for constructing an access control policy out of simpler policies and show that their algebra can be augmented to give a policy definition

language. They analyze their language's expressiveness with respect to first order logic and show that the problem of expression containment is decidable for a class of policy expressions in their language.

Winsborough et al. [20] propose a mechanism called "trust negotiation" which, given an access control policy, provides an efficient way for domains to exchange credentials to obtain a given access. However, they do not address the question of how the access control policy was decided upon. Further, their model is also of the client-server type, where each resource is owned by a single domain, and the other domains request access.

Shands et al. [18] propose an architecture for dynamic coalitions which allows for distributed enforcement of a coalition access control policy. They discuss the relationships between the policy enforcement points in the various domains and the requirements for disseminating coalition policies to these enforcement points. Gligor et al. [12] discuss the negotiation of coalition policies among peer domains. Khurana [15] proposes a mechanism and language for negotiating coalition access control policies, and also discusses the question of jointly-owned resources. However, none of these discusses the design of the negotiating agent itself.

3.2 Soft Constraints

Constraint solving has been an active area of research in Artificial Intelligence. A Constraint Satisfaction Problem (CSP, [16]) consists of a set of problem variables, a domain of possible values for each variable, and a set of constraints, each of which specifies an acceptable combination of values for one or more of the problem variables. Therefore in a CSP, each constraint is simply a set of tuples over some subset of the problem variables. A solution for a CSP is an assignment of values to the variables that satisfies all the constraints of the problem.

Research in AI has largely focused on finite CSPs, where the domain of each variable is a finite set. A number of techniques have been used for solving CSPs, including backtracking, branch-and-bound, backjumping, forward checking and arc consistency checking [16]. CSPs have been used to represent many problems, such as machine vision, map coloring, production scheduling and VLSI design.

CSPs cannot efficiently model soft constraints, which express preferences, or prioritized constraints, where certain constraints can be sacrificed if the solution is good enough with regard to some other criterion. Also, CSPs cannot model partial knowledge, where some or all constraints are unknown or partially known when the solution is computed. Semiring-based CSPs (SCSPs, [3, 5]) are an extension of CSPs wherein the constraints are not Boolean but defined over an appropriate semiring. By using a specific class of semirings which can naturally express partial orders, SCSPs address all the above shortcomings of CSPs.

A semiring is a tuple $< A, +, \times, \mathbf{0}, \mathbf{1} >$ where

- A is a set with $\mathbf{0}, \mathbf{1} \in A$;
- $+$, the additive operation, is closed, commutative and associative over A with $\mathbf{0}$ as its identity element;
- $\times$, the multiplicative operation, is closed and associative over A with $\mathbf{1}$ as its identity element and $\mathbf{0}$ as its absorbing element;
- $\times$ distributes over $+$.

A c-semiring (or constraint semiring) is a semiring such that $+$ is idempotent, $\times$ is commutative, and $\mathbf{1}$ is the absorbing element of $+$. The $+$ operation of a c-semiring then naturally defines a partial order over the elements of the semiring; if $S =< A, +, \times, \mathbf{0}, \mathbf{1} >$ is a c-semiring with $a, b \in A$ and $a + b = b$ then we say that $a \leq_S b$, which means that b is better than a under this partial order over S. It is easily shown that both $+$ and $\times$ are monotone on the ordering $\leq_S$.

An lc-semiring is a c-semiring for which A is finite and the $\times$ operation is idempotent.

A semiring-based constraint system is a tuple $< S, D, V >$ where S is a semiring, D is a finite set and V is an ordered set of variables. A constraint over such a system is a tuple $< def, con >$ where $con \subseteq V$ is known as the type of the constraint, and $def : D^k \rightarrow A$ (where k is the cardinality of V) is the value of the constraint. Thus def assigns a value from the semiring to each combination of values of the variables in con. This value can be interpreted as a strength of preference, a probability, a cost, or something else depending on the problem. An SCSP is then a tuple $< C, v >$ where $v \subseteq V$ and C is a set of constraints.

Given two constraints $< def_1, con_1 >$ and $< def_2, con_2 >$ over the above constraint system, their combination is defined as $< def, con >=< def_1, con_1 > \otimes < def_2, con_2 >$ where

- $con = con_1 \cup con_2$
- $def = def_1(t \downarrow_{con_1}^{con}) \times def_2(t \downarrow_{con_2}^{con})$, where $t \downarrow_{con_i}^{con}$ denotes the part of tuple t corresponding to variables in con_i.

Since the $\times$ operation is monotone in $\leq_S$, we can see that adding constraints can never increase the value associated with any tuple t.

If $c =< def, con >$ is a constraint over a constraint system defined as above, and $I \subseteq V$ is a set of variables, then the projection of c over I, denoted $C \Downarrow_I$, is the constraint $< def', con' >$ over the same constraint system with

- $con' = I \cap con$

- $def'(t') = \sum_{\{t | t\downarrow^{con}_{I\cap con} = t'\}} def(t)$

The solution of an SCSP is the constraint obtained by combining all the constraints in the SCSP and projecting it over the set v of variables of interest. The best level of consistency (blevel) of the SCSP is the projection of the solution over the empty set. Thus the blevel represents the highest valuation that can be attained by a tuple under the constraints. In other words, the blevel gives the maximum extent to which a given set of constraints can be satisfied. Finding the best level of consistency is an NP-complete problem, as is solving the SCSP. However, many special cases can be solved efficiently. For lc-semirings, local consistency algorithms yield approximate solutions efficiently [3]. In some special cases (where $\times$ is not necessarily idempotent), dynamic programming yields a solution in $O(n)$ time [3]. For many problems, Russian Doll Search, which combines dynamic programming and branch-and-bound techniques, converges quickly in practice [19].

SCSPs have also been used in a variety of applications. For instance, Bella and Bistarelli [2] used them to model the Needham-Schroeder protocol and showed that the model can be used to "discover" a well-known attack on this protocol.

Constraint Logic Programming (CLP, [14]) incorporates the notion of constraints into Logic Programming, by replacing term equalities with constraints, and unification with constraint solving. This allows much more concise representation of problems; it also allows for more efficient implementations of constraint solvers, as it provides additional information that helps guide the search for a solution.

Semiring-based CLP (SCLP) generalizes CLP to soft constraints. The syntax and semantics of SCLP programs are described in [4]. Briefly, an SCLP program consists of a set of clauses of the form $H :- B$. We say this clause holds in an interpretation I iff for any ground instantiation of H, say $H\theta$, we have $I(H\theta) \geq_S I(\exists B\theta)$.

A number of CLP solvers are available. A useful programming language for SCLP is $clp(FD,S)$ [11], which is based on an extension to Prolog; software tools are freely available on the web [10]. The language is restricted to lc-semirings, and uses local consistency techniques for efficiency.

4 Building a Negotiating Agent

The central idea of this paper is that a domain's access control constraints can be written as a set of constraints in the SCSP framework. Equivalently, these constraints can be formulated as an SCLP. This is not surprising, since access control constraints that are expressed in First Order Logic can be translated to logic programs in languages such as Prolog, and SCLP is a generalization of Logic Programming. However, this simple idea has powerful consequences - in particular, it leads to an elegant method for solving the negotiation problem.

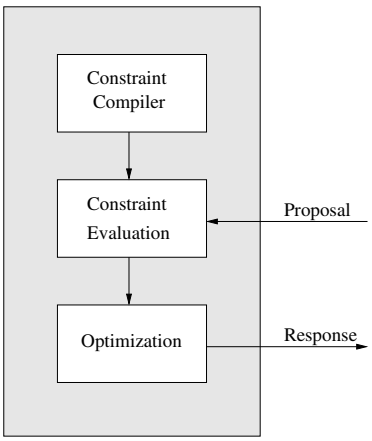

Figure 2. Structure of negotiation agent.

We start by observing that given a set of domains, each with its access control constraints and preferences expressed as an SCSP, the most desirable coalition state can be found by combining all the domain SCSPs into a single SCSP and finding a maximal (i.e. highest valued) solution to this large SCSP. However, the resulting SCSP can be very large and hard to solve. The domain SCSPs may also be large, and collecting them in one place may not be practical. Also, we would like a process that allows administrators to specify some basic constraints to start the negotiation process, and then fine-tune it by adding more constraints as they are needed.

A simple solution is to implement a round robin negotiation protocol: domains take turns proposing coalition states, and other domains can either accept or make a counter-proposal. A coalition state is committed when it is agreed to by all domains.

A sketch of our negotiation agent is shown in Figure 2. The agent consists of three main parts: a constraint compiler, a constraint evaluator and an optimizer. These may be thought of as different interfaces to a common underlying SCLP solver. The constraint compiler translates the domain's policy constraints, as well as additional constraints and preferences specified by the administrator, into an SCLP. The constraint evaluation engine takes a proposed coalition state and translates it into a goal clause which it then evaluates against this SCLP. The optimizer finds a maximal solution to the current policy SCLP.

The agent also has an interface to the administrator, which is not shown in Figure 2. The nature of this interface depends on how much human interaction is desired. If completely automatic negotiation is the goal, then the interface simply asks for the administrator to specify the coali-

tion mission and the domain preferences, and returns the outcome of the negotiation. If a greater degree of human involvement is desired, the agent can be used as a decision support tool. As an instance of this approach, the administrator could be shown each proposal made, and its value according to the constraint evaluator. She could then accept it or add more constraints to guide her agent's next proposal.

The administrator may have an additional interface to the constraint evaluator for running checks against a proposed state. For example, an administrator might want to run some sanity checks on a proposed state before accepting it. Such checks are important in practice, since specifying the preferences has been left to a human administrator and is hence an error-prone task. Since running such checks also involves checking an appropriate goal clause against the domain's SCLP, the constraint evaluator can easily be extended to handle it.

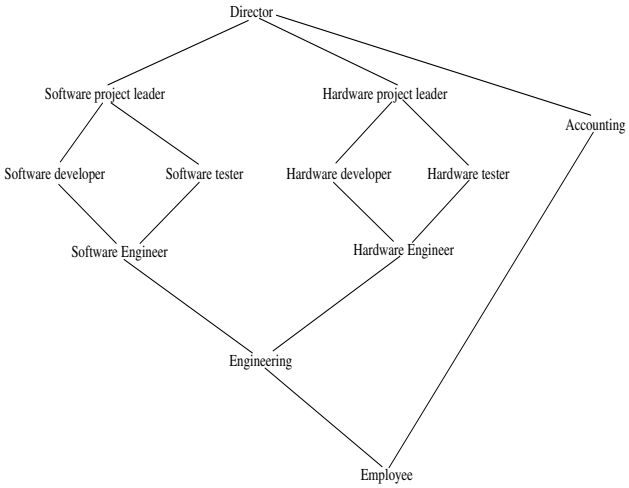

Figure 3. Example of a role hierarchy.

5 Expressing Negotiation problems as SCSPs

5.1 Expressiveness of SCLP Framework

We start by showing that the semiring-based constraint logic programming framework is expressive enough to represent a large class of interesting policies. In particular, it is easy to show that the SCLP framework is at least as expressive as RCL2000 [1].

Proposition 1 *Any set of RCL 2000 constraints can be expressed as an SCLP.*

Proof: Any set of RCL 2000 constraints can be translated into RFOPL [1]. Any RFOPL program can be written in Prolog, which is a subset of CLP. A CLP program is just an SCLP over the semiring $< \{True, False\}, \vee, \wedge, False, True >$. ■

While this result establishes the expressiveness of SCLP, it does not mean that constructing an SCLP in this way is a particularly good idea. The above transformation does not take advantage of the power of the SCLP framework at all; to do that one would have to choose a particular semiring that allows us to express the policy more economically while also providing a quantity for optimization. Translations to SCLP are also much harder to obtain for more complicated languages like Ponder and Tower.

5.2 Choosing a Semiring

The choice of semiring is important, and has a substantial effect on the economy of problem representation, as well as on the complexity of the resulting SCLP. How to choose an optimal semiring in general is an interesting open problem. However, in an RBAC system, two partial orders are naturally present: the hierarchy of roles and the hierarchy of permissions. We use these to derive semirings and use them to illustrate our framework.

An example of a role hierarchy is shown in Figure 3. Note that higher roles inherit all the privileges of their descendants, while lower roles inherit all the users of their ancestors in the hierarchy. We assume that there is always a role R_∞ that is the ancestor of all other roles (this is typically an administrator role), and that there is a role R_0 that is a descendant of all roles (this is typically a "Guest" or "User" role, and is present, at least implicitly, in all RBAC systems).

We then define the Role Semiring S_R as $< R, +_R, \times_R, R_\infty, R_0 >$, where

- R is the set of roles in the system.
- The $+_R$ operation is defined as follows: $(R_1 +_R R_2)$ is the highest common descendant of roles R_1 and R_2 in the role hierarchy.
- The $\times_R$ operation is defined as follows: $(R_1 \times_R R_2)$ is the lowest common ancestor of roles R_1 and R_2 in the role hierarchy.
- R_0 and R_∞ are as defined above.

Note that each node in the hierarchy is both an ancestor and a descendant of itself. It is easy to see that both $+_R$ and $\times_R$ are idempotent. Therefore, since R is finite, S_R is an lc-semiring.

Note that under this definition, more privileged roles are assigned lower semiring values. Therefore solving an SCSP over the Role Semiring gives us the lowest role in the hierarchy that satisfies all the constraints of the problem. Exchanging the roles of the $+_R$ and $\times_R$ operations and making

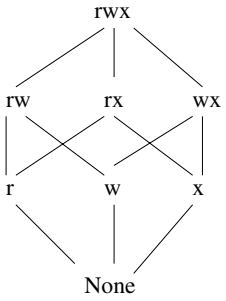

Figure 4. Example of a permissions hierarchy.

R_0 the zero element and R_∞ the absorbing element gives a new lc-semiring, which we will call the Roles$^-$ Semiring; any SCSP over this semiring gives the highest role in the hierarchy that satisfies all the constraints.

Another obvious candidate for the semiring is the permissions hierarchy. In many access control systems, the permissions hierarchy forms a total lattice, which means there is a permission P_∞ that dominates all other permissions and a permission P_0 that is dominated by all other permissions. For example, a permission hierarchy that uses the standard UNIX permission bits is shown in Figure 4.

We can therefore define the Permissions Semiring S_P as $< P, +_P, \times_P, P_\infty, P_0 >$, where

- P is the set of permissions in the system.
- The $+_P$ operation is defined as follows: $(P_1 +_P P_2)$ is the highest permission that is dominated by both P_1 and P_2.
- The $\times_P$ operation is defined as follows: $(P_1 \times_P P_2)$ is the lowest permission that dominates both P_1 and P_2.
- P_0 and P_∞ are as defined above.

Once again we follow the convention that every permission is dominated by itself. We see that both $+_P$ and $\times_P$ are idempotent. Therefore, since P is finite, S_P is an lc-semiring. Analogous to the Roles$^-$ Semiring, we can define a Permissions$^-$ Semiring by exchanging the additive and multiplicative operations of the semiring.

In more complex problems such as those with multi-domain peer-to-peer networks, we often have multiple criteria for measuring the goodness of a coalition state, and we need to do as well as possible on all criteria. To model such situations, it is useful to define more complicated semirings, and the following result is useful.

Proposition 2 *If S_1, S_2, ... S_n are semirings, with $S_i =< A_i, +_i, \times_i, 0_i, 1_i >$ and if $A = A_1 \times A_2 \times ... A_n$ (i.e. each element of A is a vector with the ith position occupied by an element of A_i), then $S =< A, +_S, \times_S, 0_S, 1_S >$ is also a semiring, where*

- $0_S =< 0_1, 0_2, ... 0_n >$
- $1_S =< 1_1, 1_2, ... 1_n >$
- *If $a =< a_1, a_2, ... a_n > \in S$ and $b =< b_1, b_2, ... b_n > \in S$ then $a +_S b =< a_1 +_1 b_1, a_2 +_2 b_2, ... a_n +_n b_n >$*
- *If $a =< a_1, a_2, ... a_n > \in S$ and $b =< b_1, b_2, ... b_n > \in S$ then $a \times_S b =< a_1 \times_1 b_1, a_2 \times_2 b_2, ... a_n \times_n b_n >$*

Further, if S_1, S_2, ... S_n are lc-semirings, S is an lc-semiring.

Proof: Follows from the semiring properties of S_1, S_2, ... S_n. ■

5.3 Formulating the SCSP

In practice, domain constraints can often be translated quickly to an SCLP language such as $clp(FD, S)$, and the solver for that language can be used to generate proposed coalition states. However, in this section we focus on the formulation of the underlying SCSP, in the hope that it will provide more insight.

In general, the procedure for formulating and solving a problem in the SCSP framework is as follows:

- Choose an appropriate semiring.
- Collect all the constraints of the problem and represent them as constraints over the chosen semiring.
- Run the solver to obtain the solution or blevel of the SCSP, as required.

The process is generally fairly straightforward. We illustrate with some examples, each of which illustrates an aspect of the negotiation problem.

Example 1 Web services - login. *A web server offers clients the ability to access a service remotely. It does this by assigning remote clients to local roles depending on credentials supplied by them. A remote client has supplied a set of credentials C^*. The web server needs to deduce which role he is assigned to.*

In this case, the Roles Semiring is a natural choice. We pick the constraint system $< R, V, C >$ where R is the Roles Semiring, C is the set of credential types known to the system, and V is the set of values these credential types can take. Then, the server's authorization policy can be expressed as an SCSP - for each rule that specifies a role r to be assigned when presented a certain set of credentials c, add a constraint that assigns the value r to the tuple c and R_0 to all other tuples. The solution to this SCSP is a constraint.

The value of the tuple C^* under this constraint is the role to be assigned to the user.

In the SCLP framework, each rule in the authorization policy would be represented as a clause. The credentials C^* would be facts added to this SCLP, and the solution gives the role to be assigned. ■

Example 2 Web services - access rights. *A web server offers clients the ability to access a service remotely. It does this by assigning remote clients to local roles. The service needs access to a number of objects (data and applications) to run. What is the minimum local role required to run the service?*

Once again, the Roles Semiring is the obvious choice. We choose the constraint system $< R, P, O >$ where R is the Roles Semiring, P is the set of permissions and O is the set of all objects required by the application to run.

Consider the server's authorization policy for the required objects. For each rule in this policy that assigns a tuple t of access rights to a role r, associate value r with the tuple t of access rights. The result is an SCSP. The value of the required access rights under this SCSP is the lowest role that is required to access the service.

For the SCLP equivalent, we translate the authorization policy to SCLP clauses, then assert the required rights as facts and find the solution. ■

Example 3 Web services redux. *A server offers clients the ability to access a service remotely. The service needs access to a number of other objects (data and applications) to run. The remote user has supplied a set C of credentials. The server uses RBAC, and must decide if the user should be allowed to access the service remotely.*

This combines Example 1 and Example 2. Solve those two problems, let the solutions be R_1 and R_2 respectively. Then if R_1 dominates R_2 the user is allowed, otherwise not. ■

The preceding examples illustrate a general method of checking that a certain policy satisfies a given set of higher-level constraints. The approach involves solving two separate SCLPs over two different constraint domains with the same semiring. The idea is to use the semiring to find the highest level of privilege granted by the policy in question. We then use the semiring to find the highest level of privilege granted when all the constraints are applied. By comparing the two we can tell whether or not the policy violates the constraints.

Example 4 Multi-domain coalition. *A number of autonomous domains (more than two) wish to collaborate on a mission. They each have their own access control constraints, and they need to run some common applications. They need to create a shared workspace for the coalition, which requires all domains in the coalition to have certain access rights to a set of shared objects. The domains have agreed upon a set of domain-wide roles, to which they assign certain domain-local roles. They need to check whether a certain assignment of domain-local roles to coalition roles is sufficient to achieve the shared workspace.*

This is a much more complicated case, and we need to use a more complex semiring (see Proposition 2). We use a composite lc-semiring based on the Permissions Semiring. Each member of this semiring is a tuple whose elements represent the domain common permissions on each of the coalition's shared objects. The constraint system is $< P, 2^L, R >$ where P is the semiring described, R is the set of domain-wide roles and L is the set of domain-local roles.

- Proceeding as in Example 1, use the SCSP induced by the domains' assignments of permissions to local roles to find the permissions associated with this particular assignment of global roles to local roles. Call it P_1.
- Proceeding as in Example 2, build an SCSP describing the access rights required to use the shared workspace. Call the resulting set of privileges P_2.
- If P_1 dominates P_2 then the shared workspace is achieved, otherwise not. ■

6 Conclusion and Future Work

In this paper we have taken some initial steps toward formalizing a mathematical framework for negotiating a common access control state between multiple domains in a peer-to-peer network. The techniques shown here are promising, but many issues remain for future research. The question of how best to choose a semiring for the problem has not been answered. For instance, we have not addressed the issue of how best to formulate the SCLP when each constraint is assigned a priority and we want to satisfy higher priority constraints before lower priority ones. We have also not explored in any detail the expressiveness of the semiring framework with regard to access control policies. We have not characterized the SCLPs generated by our negotiation problems, and so we do not know how efficient the computations will be in general.

We have also not addressed here the case when access control policies of domains are considered sensitive. In such cases, domains must negotiate based on their beliefs or estimates of other domains' policies, and must be careful not to reveal too much of their own policies in the negotiation. This may have a substantial effect on negotiating strategy, as it introduces an element of bargaining; a domain can

use the other domains' lack of knowledge about its policies to negotiate a coalition policy more favorable to itself.

Finally, the question of designing an efficient negotiation mechanism in the general case remains open. For example, the negotiation protocol could require domains to take turns in proposing a final state for the coalition, until a state is proposed that all domains agree upon. Another possibility is to allow domains to propose states in any order, while requiring that any domain not accepting a proposal must make a counter-offer. More complicated mechanisms are possible, such as protocols allowing for partial acceptance of proposed states.

Acknowledgments

We are very grateful to the anonymous reviewers for their insightful comments.

This research effort was sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-00-2-0510. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government.

References

[1] G. Ahn and R. S. Sandhu. Role-based authorization constraints specification. *ACM Transactions on Information and System Security (TISSEC)*, 3(4):207–226, 2000.

[2] G. Bella and S. Bistarelli. SCSPs for modelling attacks to security protocols. In *Principle and Practice of Constraint Programming - CP2000 Workshop on Soft Constraints, Singapore*, 2000.

[3] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint solving and optimization. *Journal of the ACM*, 44(2):201–236, March 1997.

[4] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint logic programming: Syntax and semantics. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 23(1):1–29, 2001.

[5] S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, and H. Fargier. Semiring-based CSPs and valued CSPs: Frameworks, properties and comparison. *Constraints*, 4(3), 1999.

[6] P. Bonatti, S. de Capitani di Vimercati, and P. Samarati. An algebra for composing access control policies. *ACM Transactions on Information and System Security (TISSEC)*, 5(1):1–35, 2002.

[7] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder policy specification language. In *International Workshop on Policies for Distributed Systems and Networks (POLICY 2001), Bristol, U.K.*, volume 1995 of *Lecture Notes in Computer Science*, pages 18–38. Springer Verlag, January 2001.

[8] R. Dechter. On the expressiveness of networks with hidden variables. In *Proceedings of AAAI '90*, 1990.

[9] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989.

[10] Y. Georget. Yan Georget's web page: The clp(FD,S) language. http://contraintes.inria.fr/~georget/software/clp_fds/clp_fds.html.

[11] Y. Georget and P. Codognet. Compiling semiring-based constraints with $clp(FD,S)$. In M. Maher and J.-F. Puget, editors, *Proceedings of the 4th International Conference on the Principles and Practice of Constraint Programming (CP98), Pisa, Italy, October 1998*, volume 1520 of *Lecture Notes in Computer Science*, pages 205–219. Springer Verlag, 1998.

[12] V. D. Gligor, H. Khurana, R. K. Koleva, V. G. Bharadwaj, and J. S. Baras. On the negotiation of access control policies. In B. Christianson et al., editors, *Proceedings of the 9th International Security Protocols Workshop, Cambridge, U.K., April 2001*, volume 2467 of *Lecture Notes in Computer Science*, pages 188–201. Springer Verlag, 2002. Also see transcript of discussion, pp. 202-212.

[13] M. Hitchens and V. Varadharajan. Tower: A language for role based access control. In *International Workshop on Policies for Distributed Systems and Networks (POLICY 2001), Bristol, U.K.*, volume 1995 of *Lecture Notes in Computer Science*, pages 88–106. Springer Verlag, January 2001.

[14] J. Jaffar and M. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[15] H. Khurana. *Negotiation and Management of Coalition Resources*. PhD thesis, University of Maryland, 2002.

[16] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, 1995.

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

[18] D. Shands, R. Yee, J. Jacobs, and E. J. Sebes. Secure virtual enclaves: Supporting coalition use of distributed application technologies. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, San Diego, California, 3-4 February 2000.

[19] G. Verfaillie, M. Lemaitre, and T. Schiex. Russian doll search for solving constraint optimization problems. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96), Portland, OR*, pages 181–187, 1996.

[20] W. Winsborough, K. E. Seamons, and V. E. Jones. Automated trust negotiation. In *DARPA Information Survivability Conference and Exposition (DISCEX '2000)*, January 2000.