A Framework for 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

Our research examines 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 describe 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 ever 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. Collaborative computing in peer-to-peer networks 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 and 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.

We examine the problem of automating the negotiation of access control policies between autonomous security domains. 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, and this policy will then be implemented (perhaps subject to a human operator's approval).

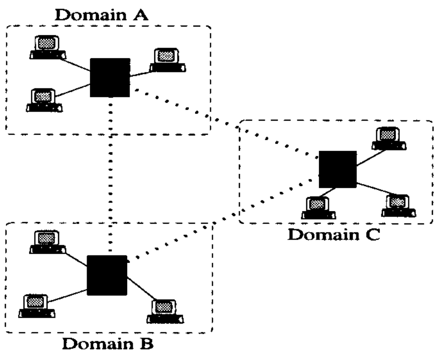

Figure 1. Negotiation in multi-domain networks.

We assume that each domain has its own access control policies, and that the negotiated state must at least satisfy the access control policies 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 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 so that a human administrator can make an informed decision.

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 have to make do with the limited knowledge gained during the negotiating process. They must also avoid revealing any more information than strictly necessary about their respective domain policies.

Even in situations where no policies are sensitive, the negotiation problem is a hard one. If each policy consists only 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 are not considered sensitive. We confine ourselves to Role Based Access Control (RBAC, [13]) systems, though our methods can be generalized to other access control models. We show that RBAC constraints, which are derived from higher-level security policies, can be expressed as constraints over an appropriate semiring. We sketch some examples and briefly compare the expressiveness of our formulation with RCL 2000 [1].

2 Background

2.1 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. 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 there is some extra-technological reason for domains to want to cooperate.

Role Based Access Control (RBAC) is a widely used model for access control systems. RBAC introduces the role as the semantic concept around which access control policy is formulated. A role typically represents one aspect of a domain's regular functions and brings together a transitory collection of users and permissions to fulfill this function. A complete RBAC model [13] includes the following state variables:

- The sets U (users), R (roles), P (permissions) and S (sessions).
- $PA : R \to 2^P$, a mapping from roles to their associated permissions.
- $UA : U \to 2^R$, a mapping from users to their roles.
- $user : S \to U$, a mapping from sessions to the respective users.
- $roles : S \to 2^R$, 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".

A number of languages have been proposed for expressing RBAC policies efficiently and unambiguously. RCL2000 [1] is a language for specifying constraints on roles in an RBAC system. The authors show that RCL2000 is sound and complete with respect to a restricted subset of First Order Logic. 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.

If a domain uses RBAC, a natural approach to sharing its resources with users from foreign domains is to set up a mapping to assign the foreign users to local roles in the domain. Herzberg et al. [9] present a language for expressing the policies to carry out this mapping, and implement their policy engine as a library and as a web server extension. However, their work is based on a server-client model, and does not extend easily to peer-to-peer networks of autonomous domains.

Winsborough et al. [15] 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. Their model is also of the client-server type. They do not address the question of how the access control policy was decided upon.

Shands et al. [14] 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. [8] discuss the negotiation of coalition policies among peer domains. Khurana [11] proposes a mechanism and language for negotiating coalition access control policies, and also discusses the question of jointly-owned resources. However, the design of the negotiating agent is not discussed.

2.2 Soft Constraints

Constraint solving has been an active area of research in Artificial Intelligence. A Constraint Satisfaction Problem (CSP, [12]) consists of a set of problem variables, a domain of possible values for each variable, and a set of constraints. Each constraint is a set of tuples representing acceptable combinations of values for 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 CSPs where the domain of each variable is a finite set. Solution techniques include backtracking, branch-and-bound, backjumping, forward checking and arc consistency checking [12]. CSPs have been used to represent many problems, such as machine vision, map coloring, production scheduling and VLSI design.

Semiring-based CSPs (SCSPs, [5, 3]) 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 subsume all the above extensions 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 $+$. An lc-semiring is a c-semiring for which A is finite and the $\times$ operation is idempotent. The $+$ operation of a c-semiring 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 $a \leq_S b$, i.e., b is better than a under this partial order over S. It is easily shown that both $+$ and $\times$ are monotone on the ordering $\leq_S$.

A semiring-based constraint system is defined as 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 $def : D^k \to A$ (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

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

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. 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. In some special cases (where $\times$ is not necessarily idempotent), dynamic programming yields a solution in $O(n)$ time [3]. For many problems, branch-and-bound techniques converge quickly in practice.

Bella and Bistarelli [2] used SCSPs 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, [10]) 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, [4]) generalizes CLP to soft constraints.

A number of CLP solvers are available. A useful programming language for SCLP is *clp(FD,S)* [7], which is

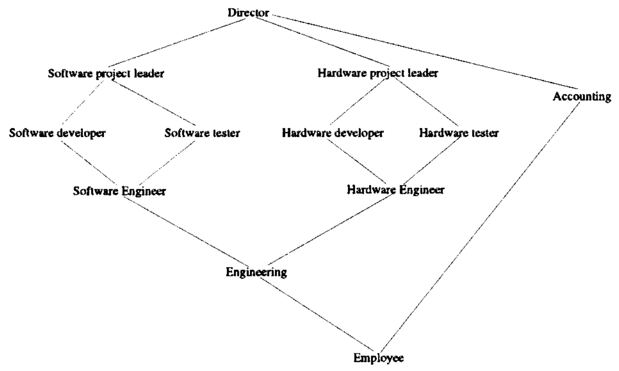

Figure 2. Example of a role hierarchy.

based on an extension to Prolog. The language is restricted to lc-semirings, and uses local consistency techniques for efficiency.

3 Expressing Negotiation problems as SCSPs

RBAC constraints can be expressed as SCSPs, which can then be solved using SCLP methods. This section outlines the process.

3.1 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. Choosing 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 2. 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.

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 R_0 the zero element and R_∞ the absorbing element gives a new lc-semiring, which we will call the Role$^-$ 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 R_∞ that dominates all other permissions and a permission P_0 that is dominated by all other permissions. 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 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, it is useful to define more complicated semirings, and the following result is useful.

Proposition 1 *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 occupies 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. ■

3.2 Formulating and Solving the SCLP

In practice, the SCLP can be formulated and solved using an SCLP language like $clp(FD,S)$ [7]. In this section we will describe SCLP formulations for some example applications.

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

- Collect all the constraints of the problem and represent them as constraints over an appropriate semiring.
- Write the above SCSP in an SCLP language.
- Run the solver to obtain the solution or blevel of the SCSP, as required.

Example 1 *Web services. A server offers clients the ability to run an application remotely. The application needs access to a number of other objects (data and applications) to run. The remote user has supplied sufficient credentials to be authenticated to role R_*. The server uses RBAC, and must decide if the user should be allowed to run the application remotely.*

In this case, we need to assign the remote user to an existing local role, therefore the Roles Semiring is a natural choice. We pick 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.

- (Initial SCSP) Express the domain's access control state as an SCSP. Note that due to our choice of constraint system, we assign roles to different configurations of permissions on the objects. So to each tuple of permissions on objects, we assign the least powerful role that has at least those permissions.
- (Application Constraints) List the operations the application needs to perform, and for each operation, add a constraint to the SCSP which assigns the value R^* to the set of permissions required for that operation.
- Find the blevel of the resulting SCSP. Call it R_{min}.
- If R_{min} is dominated by R_* then grant the request. Otherwise deny the access request.

Thus the above algorithm gives us a way to tell whether a given set of credentials is sufficient to run a given application.

Example 2 *Multi-domain coalition. A number of autonomous domains wish to collaborate on a mission. They each have their own access control policies, 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. 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 $< P,R,L >$ where P is the semiring described, R is the set of domain-wide roles and L is the set of domain-local roles. The domains then proceed as follows:

- (Initial SCSP) Express the domain policies as an SCSP. To each assignment of users to roles, assign the semiring value that reflects the shared permissions.
- (Sharing constraints) For each object that is required to be shared, assign the corresponding semiring value to the tuple that represents the negotiated role assignment.
- Find the blevel of the SCSP. If this is the same as the desired permission set for the shared workspace, then the role assignment was correct, otherwise not.

3.3 Expressiveness of SCLP Framework

The semiring-based constraint logic programming framework is expressive enough to represent a large class of interesting policies. In particular, it is at least as expressive as RCL2000 [1].

Proposition 2 *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 (such as the Role Semiring or the Permissions Semiring) that allows us to express the policy more economically while also providing a quantity for optimization.

4 Research Directions

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 must be addressed in any practical implementation. For instance, we need to address 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 are also exploring the expressiveness of the semiring framework with regard to access control policies. We are trying to characterize the SCLPs generated by our negotiation problems, in order to quantify the computational complexity of our approach.

In military coalition scenarios, access control policies of domains are often considered sensitive information. 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

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] 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.

[8] 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.

[9] A. Herzberg, Y. Mass, J. Michaeli, D. Naor, and Y. Ravid. Access control meets public key infrastructure, or: Assigning roles to strangers. In *Proceedings of IEEE Symposium on Security and Privacy*, Oakland, California, May 2000.

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

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

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

[13] 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.

[14] 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.

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