Context-Sensitive Access Control for Open Mobile Agent Systems

Christine Julien, Jamie Payton, and Gruia-Catalin Roman
Department of Computer Science and Engineering
Washington University in St. Louis
{julien, payton, roman}@wustl.edu

Abstract

The increased pervasiveness of wireless mobile computing devices draws new attention to the need for coordination among small networked components. The very nature of the environment requires devices to interact opportunistically when resources are available. Such interactions occur unpredictably as mobile agents generally have no advance knowledge of other agents they will encounter over the lifetime of the application. In addition, as the ubiquity of communicating mobile devices increases, the number of application agents supported by the network grows drastically. Managing access control is crucial to such systems, and application agents must directly manipulate and examine access policies because the agents require full control over their data. However, because these networks are often decoupled from a fixed infrastructure, reliance on centralized servers for authentication and access policies is impractical. In this paper, we explore the essential features of general access control policies tailored to the needs of agent coordination in the presence of physical and logical mobility. This access mechanism derives much of its flexibility and expressiveness from its ability to take into account context information. We propose and evaluate novel constructs to support such policies, especially in the presence of large numbers of highly dynamic application agents.

1 Introduction

Ubiquitous computing devices communicate wirelessly, opportunistically forming ad hoc networks not connected to a wired infrastructure. These networks can include a handful of devices or thousands of heterogeneous components, making coordinating and mediating their competing needs a massive task. In such environments, distributed applications exchange information or coordinate tasks. These applications are commonly structured as a logical networks of numerous application agents, and much research focuses on developing middleware to facilitate interactions among these highly dynamic application components.

This paper focuses on systems that use tuple spaces as a basis for coordination among mobile application agents. The original Linda model [6] provides a centralized tuple space where application agents exchange information using content-based matching of patterns against data. Variations on this theme adapt it to the mobile environment where a central repository is not feasible. Instead, applications residing on physically mobile hosts exchange information via a shared, distributed variation of the traditional Linda tuple space. The benefits of using this tuple space model are twofold. First, the tuple space affords a highly decoupled manner of communication, eliminating the need for a priori knowledge of the identities of communication partners. This facilitates flexible coordination in open environments in which mobile agents come and go without notice. Second, use of the model masks the complex communication details associated with handling the frequent, unannounced disconnections that characterize ad hoc networks. A number of mobile agent middleware systems designed for ad hoc networks realize this style of tuple space coordination, including LIME [11], EgoSpaces [9], MARS [4], and TuCSoN [13]. These systems mask the complex communication details of the mobile ad hoc network setting and support the rapid development of applications in such environments.

Due to the open and dynamic nature of such mobile systems, security concerns of three types arise: protecting mobile hosts from malicious agents, protecting agents from tampering hosts, and securing data. Several approaches addressing the first two concerns exist for mobile agent systems. For example, D'Agents [7] uses public-key cryptography to authenticate incoming agents to increase host security. Undetachable threshold signatures [1] prevent hosts from tampering with

an agent's data.

Protecting data includes both ensuring data integrity and controlling access. Much coordination research has addressed the former by encrypting communication within coordination spaces. SAMCat [12] and Yalta [3] use encryption and authentication to securely transmit tuples into and out of a data space. Our work focuses on the final issue: controlling data access. A solution to this problem is complicated by the fact that, in the mobile environment, disconnection from a wired infrastructure renders a centralized solution impossible.

In traditional solutions to access control, mediating access to objects in a system is the task of a single administrator who determines what kind of access can be provided to particular subjects for certain objects. A common mechanism for addressing access control in wired networks uses access matrices to describe rights. The rows of the matrix correspond to users and the columns to objects; a cell in the matrix contains the access rights a user has on an object. This approach generalizes several commonly used approaches, including access control lists and capability definitions. In the mobile environment, the number of possible agents and the amount of data available over the lifetime of the system makes directly applying these solutions impractical. The access control function introduced in this paper overcomes the limitations imposed by mobile systems by operating over general descriptions of interacting parties and dynamically adjusting to the changing context.

Section 2 introduces a general mobile coordination model for mobile computing. Section 3 describes our access control mechanism. Details of a particular implementation of this mechanism appear in Section 4. In Section 5, we discuss the construct's expressive power and overhead. Section 6 overviews related work, and conclusions appear in Section 7.

2 A Generalized Coordination Model

In this section, we capture the essential features of the tuple space coordination mechanisms used in mobile agent systems in order to explain access control requirements for mobile middleware. The result is a generalization that spans the gamut from tuple definition to sophisticated operations. In the original Linda model, processes generate tuples in a centralized repository and retrieve them using content-based operations in which the retrieving process specifies a pattern that the returned tuple must match. These operations are synchronous in that they "block" the issuing process until a tuple satisfies the operation and is returned.

The Tuple Space. Some mobile systems (e.g., MARS [4]) focus on logically mobile agents in a network of physically stationary hosts, while other systems (e.g., LIME [11] and EgoSpaces [9]) integrate physical and logical mobility. All such systems facilitate the interactions of large numbers of application agents by associating a tuple space with a network component that allows other components to access the data. Tuple spaces can be permanently bound to hosts, to agents, or distributed among a combination of the two. The distribution of the tuples is irrelevant with respect to access control; the key aspect of the representation is how application agents access data. In this paper, we assume a tuple space bound to each mobile agent. Using this model, we can simulate other approaches, e.g., to simulate tuple spaces bound to a host, we permanently associate an agent to each host and use its tuple space as the host's tuple space.

Tuples and Patterns. We generalize a tuple to one in which a field is identified by a name. A tuple is an unordered set of triples: $\langle(name, type, value), \ldots\rangle$. For each field, *type* is the data type of *value*. In a tuple, each field *name* must be unique. Users access tuple spaces by matching patterns against tuples. A pattern has the form: $\langle(name, type, constraint), \ldots\rangle$. A *constraint* is a function that provides requirements a field's *value* must match for the tuple's field to match the pattern's field. Specifically, the matching function $\mathcal{M}$ is defined over a tuple θ and a pattern p as:

$$\mathcal{M}(\theta, p) \equiv \langle \forall c : c \in p :: \langle \exists f : f \in \theta \wedge f.name = c.name \wedge f.type \texttt{ instanceof } c.type :: c.constraint(f.value)\rangle\rangle.$$ [1]

$\mathcal{M}$ requires that, for every constraint c in the pattern, there is a field f in the tuple with the same name, the same type or a derived type, and a value that satisfies c. While the function requires that each constraint is satisfied, it does not require that every field in the tuple is constrained, i.e., a tuple must contain all the fields in the pattern but can contain additional fields.

Basic Operations. Next, we classify the available operations, regardless of the tuple space structure.

Tuple Generation. Agents create tuples using **out** operations. Tuple generation generally places a tuple (t) in a specific tuple space: **out**(T, t), where T is a tuple space with a particular name located at a particular agent. In EgoSpaces, an **out** places the tuple in a local tuple space controlled by the generating agent.

[1] In the notation $\langle$**op** *quantified_vars* : *range* :: *exp*$\rangle$, the variables from *quantified_vars* take on all values permitted by *range*. Each instantiation of the variables is substituted in *exp*, producing a multiset of values to which **op** is applied, yielding the value of the three-part expression. If no instantiation of the variables satisfies *range*, the value of the expression is the identity element for **op**, e.g., *true* when **op** is $\forall$.

In LIME an **out** can place a tuple in any tuple space owned by any agent on a connected host. In MARS the tuple is created in the local host's tuple space.

Tuple Retrieval. To read and remove tuples, agents use **rd** and **in** operations respectively, which assume three forms: blocking, atomic probing, and scattered probing. The blocking form, **rd**(T, p), returns a tuple matching the pattern p from the tuple space T. The tuple space can be either local to the agent or controlled by another network component. Atomic probing operations, **rdp** and **inp**, guarantee, if a matching tuple exists, it is returned, but they can return ϵ if no match immediately exists. Like the blocking operations, they are atomic with respect to the tuple space on which they are issued; in some cases in the mobile environment, guaranteeing this atomicity can be expensive. Scattered probing operations, **rdsp** and **insp** offer weaker guarantees. While all of these access operations entail only single tuples, many extensions to Linda allow simultaneous access to groups of tuples. These operations come in all three forms described above and are referred to as group operations, e.g., **rdg** refers to a blocking non-destructive read operation that returns all matching tuples from the tuple space.

Different models present tuple space operations to the user in different ways. In LIME, application agents operate over a federation of connected tuple spaces, while in EgoSpaces, agents operate over projections, called *views*, of all available data. In these cases, the more complex interactions can be reduced to the tuple space operations described above.

3 Access Control Function

As dynamic components become increasingly pervasive, security concerns become of paramount importance. Given the coordination model described previously, an agent assumes responsibility for mediating access to its data. The ability to control access in this manner is fundamental because it allows the access policies to reflect an agent's instantaneous needs. This is especially important in the highly dynamic mobile environment where mobile agents want to constantly adjust their behavior to adapt to a changing context that can include communicating with unpredictable parties. To achieve flexible access control in this environment, each agent specifies an individualized access control function.

We allow an agent to restrict which other agents access its data and the manner in which the access occurs. To accomplish the former, a requesting agent must provide credentials identifying itself. To accomplish the latter, the access policy accounts for the operation being performed. In the end, each agent defines a single access control function that takes as parameters a tuple, a set of credentials identifying the requesting agent, the operation being performed, the pattern used in the operation, and the owning agent's profile (defined next). This function returns a boolean indicating whether the requested access is allowed.

Profiles. Before describing the access control function in more detail, we introduce a profile to maintain properties of each agent, which we represent as a tuple. Particular applications or coordination systems may require specific attributes in this profile. In general, we assume a profile contains at least a unique host id identifying the agent's host and a unique agent id.

Parameters. An access control function takes five parameters: the credentials, operation, tuple, pattern, and the owner's profile.

Credentials. Credentials allow an agent that is requesting access to convey information about itself. In simple cases, they can be a standard set of attributes, e.g., the agent's id or a third-party authentication. When an agent has a priori knowledge of the access requirements, credentials can be more complicated, e.g., a password. When constructing credentials, an agent must take care not to give away too much information, e.g., if the agent has multiple passwords, it should send only the correct one. This identification is especially necessary in open and dynamic mobile environments, where it is often not possible to know a priori exactly which agents can access restricted information. Instead, agents must prove they have required privileges. Credentials are a subset of the agent profile and are presented as a tuple of attributes, which allows the access control function to use pattern matching to evaluate credentials. The credentials and their transmission with the operation are assumed to be private. This security is outside the scope of this paper but could be accomplished using cryptography schemes already under development.

Operation. The access control function can also account for the operation requested. Often, some data should be restricted to read-only access, yet current systems do not inherently allow this restriction. Considering the operation when determining access allows a dynamic application to permit one set of operations for some agents, but different operations for others.

Requested Tuple. Because we focus on tuple space models, the access control function can operate over the tuple to be returned from an operation. Pattern-matching allows this portion of the access control function to be easily defined while remaining flexible.

Pattern. A powerful component of the access control function is its ability to account for the pattern used

in the content-based operation. The pattern provides information about an application's prior knowledge of the data. The owning agent may allow access only to agents that know the "correct" way to access the data (e.g., providing a wild card pattern that matches any tuple may not be acceptable). Some knowledge of the structure of the requested tuple might indicate that the requesting agent shares common application goals.

Owner's Profile. The access control function also considers the owner's current state. Because the access policy is determined dynamically, access can be granted based on context information. In some cases, data may never be sent wirelessly between devices unless they are within a secure physical environment where eavesdropping is known to be impossible.

Access Control Function. The access control function takes the five parameters described above, and determines whether or not to allow the requested access. Formally, this function can be represented as: $\text{ACF} : T \times C \times O \times P \times \Pi \rightarrow \{0, 1\}$, where T is the universe of tuples, C is the universe of credentials, O is the finite set of operations, P is the universe of patterns, and Π is the universe of profiles. The access control function (ACF) maps the values of the parameters to a boolean indicating the access decision. The function can also be represented as: $access = \text{ACF}(credentials_r, op, tuple, pattern, profile_o)$; r is the requesting agent and o is the tuple's owner.

We will briefly discuss the expressive power of this construct later. For now we consider what it *cannot* easily represent. Access decisions cannot be based on properties of the requesting agent not included in its credentials. Therefore the requesting agent must carefully construct the credentials it sends with each operation request. Also, the access decision can also not rely on arbitrary environmental properties. For example, an agent cannot base a decision on the number of copies of a tuple.

The access control function lends itself well to the mobile environment because it allows access policies to adapt to the context. Access decisions are transparent to requesting agents; if access is denied, a requester does not even know that the matching tuple existed.

4 Implementation

We have integrated the access control function described in the previous section with the EgoSpaces coordination model and middleware. As discussed previously, access control solutions for ad hoc network settings should be distributed and not require a connection between a controller and an agent to evaluate access rights. In this section we first highlight the novel features of the EgoSpaces system that make it amenable to coordination in ad hoc networks. We then discuss the components of the access control implementation as they relate to EgoSpaces.

4.1 EgoSpaces Overview

EgoSpaces addresses the needs of agents in large-scale heterogeneous environments. An agent operates over a context that can include, in principle, all data in an entire network. EgoSpaces' unique model of coordination, however, structures data in terms of *views*, or projections of the maximal set of data. Each agent defines its own views; these individualized views abstract the dynamic environment by constraining properties of the network, hosts, agents, and data. To further reduce programming costs, EgoSpaces transparently maintains views; as hosts and agents move, the view's content automatically reflects context changes without the agent's explicit intervention. EgoSpaces employs the agent-specified access control function on a per-view basis. When an agent defines a view, it attaches a set of credentials and a list of operations it intends to perform on the view. The EgoSpaces middleware can then use each contributing agent's access control function to determine which tuples belong in the view. In the end, the view contains only tuples that qualify via their owning agent's access control function.

4.2 A Model Perspective

In providing the previously described access controls in EgoSpaces, we add to the middleware both credentials and access control functions and use the content-based retrieval and pattern matching mechanisms already present in the system. The view of an EgoSpaces agent contains a subset of the tuples present in the logically shared tuple space spanning all transitively connected agents. The contents of the view are determined by a view specification, described in [9]. Specifically, an agent's view includes only those tuples stored in its local tuple space or the local tuple spaces of connected agents that satisfy a set of constraints provided in the view specification. Upon integrating the access control function, a set of credentials is now also included as part of the view definition. These credentials are simply properties that convey information about the agent. The agent's credentials can be altered at any time during the agent's lifetime. To provide the ability to restrict the agent's perspective according to the credentials, an agent also provides a dynamically modifiable access control function. The agent's credentials are compared to the access control functions of agents

who contribute data to the view to further restrict the view according to the appropriate agents' access restrictions.

4.3 An Implementation Perspective

An agent defines a view as a set of constraints: network constraints, host constraints, agent constraints, data constraints, and, in this access control extension of the system, credentials. The first four components of the view definition are outside the scope of this work; instead we focus on the components crucial to the provision of flexible and context-sensitive access controls. Each agent also defines an access control function. The view is managed on behalf of an agent by a component in the EgoSpaces infrastructure, the `EgoManager`. Each host is associated with a single `EgoManager`, and all the agents residing on a host register with the `EgoManager` before coordinating with other agents. When registering, an agent's local tuple space contents become the responsibility of the `EgoManager`, who mediates communications between connected agents. The application agents use the `EgoManager` to define and interact with their views. An agent issues content-based retrieval operations on its views. These operations are actually serviced by the `EgoManager` with which the agent is registered. The `EgoManager` uses the pattern provided to select tuples that match the pattern provided with the operation and evaluates each tuple individually to determine whether or not the tuple satisfies the view and is a viable candidate for return to the requesting agent. In evaluating each tuple, the `EgoManager` extracts information about the agent (properties of the host the agent resides on, properties of the agent, and the agent's access control function) that is providing the tuple and compares this information with the constraints defined in the requesting agent's view, *including the credentials*. The latter is key to the access control function's integration into the EgoSpaces middleware. If the tuple satisfies the view's constraints *and* the requesting agent's credentials satisfy the tuple owner's access control function, then the tuple can be returned to the requesting agent.

An important aspect of the integration of the access control mechanism described in Section 3 into EgoSpaces revolved around the fact that it relies on the mechanisms inherent to tuple space based systems to mediate access. Tuples are used to describe credentials, and access control functions can be described by a set of access policies defined as patterns, or templates, over tuples. Implementing credentials and access control functions in this way provides a number of benefits. First, the pattern matching mechanisms already provided by the tuple space system can be used to check the credentials against an access control function. Second, we allow the programmer to construct credentials and access control functions in a way that he is already familiar with. Third, using tuples and templates allows for flexibility and adaptation, since adding and removing fields from tuples and patterns is relatively simple. Finally, the use of tuples and patterns allows for expressive access control functions and credentials. Access control may be expressed according to any property of the interacting agents, as long as the properties can be captured in tuple and template form.

4.4 A Music Sharing Application

A music sharing application for mobile users implemented on top of EgoSpaces serves as the vehicle for testing the access control implementation. The application provides users with access to a music service with sharing, search, and download capabilities.

To determine which music a user sees, the user provides properties that define the music sharing application's view. This includes a network constraint that includes only data residing on hosts within a certain number of network hops, a host constraint that requires the data to reside on hosts which are traveling in the same direction as the user, and a data constraint that restricts the returned items according to a file size limit. A screen shot of the resulting application is shown in Figure 1.

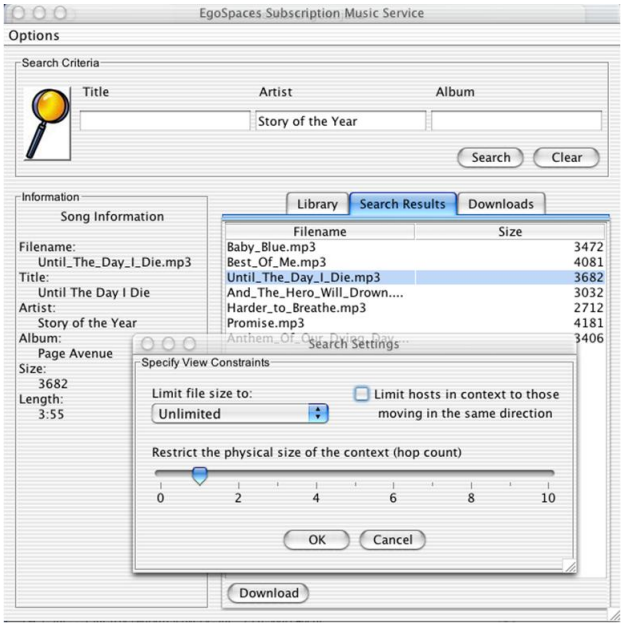

Figure 1. The subscription music service

The data is also restricted according to the creden-

tials provided by the agent, which includes a unique agent id and a known phrase encrypted with a shared password provided in the user's official registration from the music service. This password encrypted phrase authenticates the user as a subscriber. Since users share music only with others subscribed to the service, the agent also provides an access control policy which specifies that a requesting agent must have an agent id and must have the correct phrase encrypted with the subscription password. Successful decryption of the phrase by the receiving agent implies that the requesting agent holds the correct password. The code to define the credentials within the application is:

```
Credentials c =
   new Credentials(getAgentID());
c.addProperty(‘‘Passphrase’’, encryptedPhrase);
```

To build the access control policy, the agent defines the policy and adds it to the access control function:

```
AccessControlPolicy policy =
   new AccessControlPolicy();
policy.addPropertyConstraint(‘‘Passphrase’’,
    String.class,
    new EquivalencyConstraintFn(encryptedPhrase));
policy.addPermittedOperation(Operations.RDP);
acf.addPolicy(policy)
```

This example indicates that the code needed for an agent to perform access control using the implementation presented in this paper is relatively simplistic and minimal. However, this example is limited in that some centralized authority is in fact needed to distribute the passwords and phrases required to access the music sharing service and to update them as needed. Not only is this solution centralized, but it does not highlight the dynamic capabilities of the access control mechanism described in this paper. This does not mean that our access control implementation is limited in the same way. In fact, many additional application domains, including administrative domains, illustrate the distributed and dynamic nature of the credentials and access control function.

4.5 Administrative Domains

Many applications restrict agent operations to administrative domains. Assume nested domains defined as a university's computers, a department's computers, and a research group's computers. To provide security guarantees, applications limit access to certain data to only computers on the university's network. Still other data ought to be restricted to departmental computers or to research group computers. A user in the research group, working on a mobile computer, wants to use a software license of which the research group has n copies. The licenses are stored as tuples in a tuple space. Each computer in the group carries a tuple space; the available licenses are initially distributed in some random fashion. A user can take a license if it is not in use and the user holding the license is within communication range. The agents controlling the licenses restrict access to only group members who have departmental authentication (retrieved a priori), and are running on computers in the university domain. To retrieve a license, a user provides these three properties as credentials and attempts to **in** a license from a connected tuple space. If successful, the number of available licenses decreases by one. When the user finishes using the software, it replaces the license in its local tuple space.

5 Discussion

The access control function provides a flexible mechanism for agents to specify privileges dynamically and adaptively in mobile coordination systems.

Expressiveness. While its expressiveness makes the access control function more flexible and arguably more useful in coordination among constantly changing mobile agents, this flexibility comes with some cost.

Credentials. On one hand, because credentials can encode arbitrary information about an agent, particular applications can adapt credentials to their needs. On the other hand, a requesting agent must be careful not to reveal too much information since any information sent in credentials is no longer secret.

Functions. Because the access control function takes a number of parameters, an agent can dynamically adjust its access policies. Again, flexibility comes with a cost. While complex access control policies are possible, constructing the function (from the developer's perspective) can become difficult. Fortunately, the design of the function prevents this complexity from affecting agents that do not require complex policies.

Overhead. Given the model's expressiveness, it is useful to evaluate its overhead. The addition of the access control mechanism introduces some amount of programming overhead, but this overhead is difficult to quantify without a case study involving users implementing actual access control policies. While this is a useful future task, it is outside the scope of this paper. Instead we focus on the overhead due to the additional communication and computation needed to provide the access control function described previously.

Additional Communication. The key aspect of the communication overhead is the amount of data (in bits) that must be sent. Before adding the access control

mechanism, the number of bits required to send an operation request is: $b = |op| + |pattern| + |agent\_id_r|$, where $|op|$ is the number of bits required to identify the operation. $|pattern|$ is the number of bits required to represent the pattern, which depends on the number of fields in the pattern. $|agent\_id|$ is the number of bits required to identify the requesting agent so the response can be returned. It is likely that the pattern, which encodes the content-based nature of the request, dominates this expression, as the *op* and $agent\_id_r$ are simple data types with small, constant lengths.

We can write a similar term to express the number of bits needed to be sent when using the access control function. This includes only the addition of the number of bits necessary to encode the credentials: $b_{acf} = |op| + |pattern| + |agent\_id_r| + |credentials_r|$.

Credentials are a tuple. Because tuples are similar to patterns the number of bits required to represent the credentials is likely near the number of bits needed to represent a pattern. If so, the overhead of using access control is approximately 2. An application can directly control the amount of overhead it incurs because it determines what credentials to send with each request. In this respect, the use of application intuition to reduce the credentials transmitted to exactly those required reduces the overhead of the communication.

Additional Computation. Evaluating the access control function also requires additional computation in the form of an additional method invocation. Because the function can contain arbitrary code, the computational overhead lies in the hands of the application programmer. From the programmer's perspective, the operating conditions of the application must be a primary concern. If so desired, a system can include a mechanism to prevent undesirable access control functions by bounding the time they are allowed to run or by imposing restrictions on their capabilities. In most cases, however, the additional computation required is minimal since the access function may be limited to a pattern matching function.

6 Related Work

As discussed previously, the use of an access matrix does not directly lend itself to mobile systems. In one example of attempting to apply such a method, TuCSoN agents [5] are assigned capabilities defining tuple space operations for particular patterns in a certain tuple space. An access control list for the tuple space stores these capabilities. This approach requires that all coordinating parties are known in advance and that a centralized party can determine access policies statically.

Other systems use encryption for access control. In SecOS [2], for example, tuples are unordered sequences of individually encrypted fields, and, to match an encrypted field, a pattern must contain a correct key. Other work [8] associates keys with tuple spaces, and an agent must provide the key to access the tuple space. While both of these models provide access control mechanisms, they require secure key distribution and management, which affects the scalability of the system.

Law Governed Interaction (LGI) [10] provides an expressive approach to access control in which agents must adhere to a law that imposes context-sensitive constraints on the execution of tuple space operations. A law dictates actions an agent performs in response to tuple space operations. Programming applications in LGI requires programming specific actions in the access control policy and adding a controller to mediate tuple space requests. In contrast, in our model, programming takes place in the coordination model, and the agent's requested operation is checked with the access control function.

7 Conclusion

In this paper, we first provided a generalized coordination model representative of those used in dynamic pervasive computing environments. We then introduced access control functions for mobile coordination and showed how they could be successfully used in these systems. Specifically, we described how we include the access control mechanism in the EgoSpaces middleware system and discussed its use in implementing an example application. While this construct does incur some overhead, the expense is not prohibitive when compared with the benefits it offers. The novel access control function directly addresses the specific access control needs of mobile coordination models. In particular, the construct provides increased scalability and decoupling when compared with previous approaches, without sacrificing flexibility and expressiveness.

ACKNOWLEDGEMENTS

This research was supported in part by the Office of Naval Research under ONR MURI research contract N00014-02-1-0715. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the Office of Naval Research.

References

[1] N. Borselius, C. J. Mitchell, and A. Wilson. Undetachable threshold signatures. In *Cryptography and Coding—Proc. of the 8th IMA Int'l. Conf.*, volume 2360 of *LNCS*, pages 239–244, 2001.

[2] C. Bryce, M. Oriol, and J. Vitek. A coordination model for agents based on secure spaces. In P. Ciancarini and A. Wolf, editors, *Proc. of the 3rd Int'l. Conf. on Coordination Models and Languages*, pages 4–20. Springer-Verlag, 1999.

[3] G. Byrd, F. Gong, C. Sargor, and T. Smith. Yalta: A secure collaborative space for dynamic coalitions. In *IEEE 2nd SMC Info. Assurance Workshop*, 2001.

[4] G. Cabri, L. Leonardi, and F. Zambonelli. MARS: A programmable coordination architecture for mobile agents. *Internet Computing*, 4(4):26–35, 2000.

[5] M. Cremonini, A. Omicini, and F. Zambonelli. Coordination and access control in open distributed agent systems: the TuCSoN approach. In A. Porto and G.-C. Roman, editors, *Coordination Languages and Models*, volume 1906 of *LNCS*, pages 99–114. Springer-Verlag, 2000.

[6] D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[7] R. Gray, D. Kotz, G. Cybenko, and D. Rus. D'Agents: Security in a multiple-language, mobile-agent system. In G. Vigna, editor, *Mobile Agents and Security*, volume 1419 of *LNCS*, pages 154–187. Springer-Verlag, 1998.

[8] R. Handorean and G.-C. Roman. Secure servise provision in ad hoc networks. In *Proceedings of the 1st Int'l Conf. on Service Oriented Computing*. (to appear).

[9] C. Julien and G.-C. Roman. Egocentric context-aware programming in ad hoc mobile environments. In *Proc. of the 10th Int'l. Symp. on the Foundations of Software Engineering*, November 2002.

[10] N. Minsky, Y. Minsky, and V. Ungureanu. Safe tuplespace-based coordination in multi agent systems. *Journal of Applied Artificial Intelligence*, 15(1), January 2001.

[11] A. L. Murphy, G. P. Picco, and G.-C. Roman. LIME: A middleware for physical and logical mobility. In *Proc. of the 21st Int'l. Conf. on Distributed Computing Systems*, pages 524–533, 2001.

[12] National Center for Supercomputing Applications, Integrated Decision Technologies Group. SAMCat: A securable active metadata catalogue. 2002.

[13] A. Omicini and F. Zambonelli. TuCSoN: A coordination model for mobile information agents. In *Proc. of the 1st Int'l. Workshop on Innovative Internet Info. Systems*, pages 177–187, 1998.