Adaptive Access Control in Coordination-Based Mobile Agent Systems

Christine Julien[1], Jamie Payton[2], and Gruia-Catalin Roman[2]

[1] Department of Electrical and Computer Engineering
The University of Texas at Austin
c.julien@mail.utexas.edu
[2] Department of Computer Science and Engineering
Washington University in Saint Louis
{payton, roman}@wustl.edu

Abstract. The increased pervasiveness of mobile devices like cell phones, PDAs, and laptops draws attention to the need for coordination among these networked devices. The very nature of the environment requires devices to interact opportunistically when resources are available. Such interactions occur unpredictably as device users have no advance knowledge of others they will encounter. The openness of these environments also requires users to protect themselves and their data from unwanted interactions while maintaining desired, yet unscripted, coordination. As the ubiquity of communicating mobile devices increases, the number of applications supported by the network grows drastically and managing access control is crucial to such systems. Application agents must directly manipulate and examine access policies because these networks are often decoupled from a fixed infrastructure, rendering reliance on centralized servers for authentication and access policies impractical. In this paper, we explore context-aware access control policies tailored to the needs of agent coordination in open environments that exhibit mobility. 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 logical networks of mobile agents. Mobile agents (or application agents) carry all or part of a particular application's behavior and are empowered with the ability to move through the network of physically mobile devices. Much research focuses on developing middleware to facilitate interactions among these highly dynamic application agents.

This paper focuses on systems that use tuple spaces for coordination, The original Linda model [1] 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. The benefits of a tuple space model are twofold. First, the tuple space affords a 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, the model masks the complex communication details associated with handling frequent, unannounced disconnections that characterize mobile networks. This allows novice programmers to create complex applications in environments for which it is generally difficult to program.

Tuple space implementations have enjoyed much popularity not only within the research community, but also in the commercial sector, where applications have reached real-world deployed status. OptimalGrid [2] uses IBM's TSpaces [3] to coordinate parallel processes in large-scale computations. TSpaces also supports communication among devices in an automobile, among components of a smart house, and in vending machine maintenance. JavaSpaces [4] supports the Jini service infrastructure and has been deployed in many situations including the integration of proprietary law enforcement databases to enhance information availability and the creation of tourism networks linking potential travelers, airlines, and hotels. More recently, a number of mobile agent middleware systems designed for ad hoc networks have begun to utilize tuple space based coordination including LIME [5], EgoSpaces [6], and MARS [7]. These systems address tuple space coordination in highly dynamic environments.

In open and dynamic mobile systems, security concerns of three types arise: protecting hosts from malicious agents, protecting agents from tampering hosts, and securing data. Commonly referenced approaches [8] address the first two concerns in mobile agent systems. Executing agents using "safe interpreters" [9–11] provides a sandboxing effect that protects hosts from errant code. Proof-carrying code [12] can verify an agent before it runs on a new host. D'Agents [10] uses public-key cryptography to authenticate incoming agents. The more difficult problem of protecting agents from tampering hosts comes in two forms: detecting a malicious event and preventing the leakage of sensitive information. The former can be accomplished by examining execution traces while encryption schemes [13] have helped to preserve an agent's secrecy. Finally, undetachable threshold signatures [14] prevent hosts from tampering with an agent's data.

Protecting data includes ensuring secrecy and controlling data access. Much research in ad hoc networks has specifically addressed securing ad hoc routing protocols. In addition, approaches like the Secure Message Transmission protocol [15] focus on protecting individual data transmissions. Even within the coordination arena, researchers have devised encryption schemes for communication with coordination spaces. For example, SAMCat [16] and Yalta [17] use encryption and authentication to securely transmit tuples into and out of a data space. Our work focuses on the final issue: controlling access to data. 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 access control solutions, a single administrator determines what kind of access can be provided to particular subjects for certain objects. A common mechanism 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 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 make direct application of 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 coordination model for mobile computing. Section 3 describes our access control mechanism. Details of a particular implementation of this mechanism appear in Section 4 and applications showing its use in Section 5. In Section 6, we discuss the construct's expressive power and overhead. Section 7 overviews related work, and conclusions appear in Section 8.

2 A Generalized Coordination Model

In this section, we capture the essential features of tuple space coordination mechanisms in mobile agent systems. This generalization of coordination allows us to focus our efforts on creating access control that is not tailored for use in a specific system. The result is a generalization that spans the gamut from tuple definition to sophisticated tuple space operations.

2.1 Linda Tuple Space Model

Linda enables coordination through the use of a centralized data repository. Processes insert data by generating tuples in the repository and retrieve data through content-based operations on the tuple space. In such an operation, the requesting process specifies a pattern that the retrieved tuple must match. These operations are synchronous in that they "block" the issuing process until a tuple satisfies the operation. Adaptations of this model of coordination have proven useful in mediating interactions among components that require decoupling in both space and time, a characteristic of highly dynamic or mobile systems.

2.2 Computational Model

We assume a computing model in which devices (or *hosts*) can move in physical space and applications are structured as a community of mobile agents that can migrate among hosts. In our computing model, the agent is the unit of modularity, execution, and mobility, while a host is a container for agents characterized by, among other things, a location in physical space. We use the term agent to refer to any stand-alone piece of software code capable of moving between connected hosts. Communication among agents and agent migration can take place whenever the hosts involved can physically communicate with each other.

2.3 The Tuple Space

Some mobile systems (e.g., MARS [7]) focus on logically mobile agents in a network of physically stationary hosts, while other systems (e.g., LIME [5] and EgoSpaces [6]) integrate physical and logical mobility. Each of these systems facilitates interactions among large numbers of application agents by using a tuple space that other hosts or agents can access. 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. This choice is motivated simply from a modeling perspective to simplify the discussion of and reasoning about our access control policies. Using this model, we can simulate other approaches. For example, 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. On the other hand, to simulate access control policies bound to an individual data item, we can create a new agent for the individual data item. The data item's access control is then controlled by the dedicated agent.

The control of each unit (agent, host, or event data item) over its own data caters to the needs of mobile applications that must often operate autonomously in order to handle the uncertainty of the environment. Agents or devices may interact for a period of time, only to be disconnected and never meet again. Such challenges render any centralized approach to data management infeasible.

2.4 Tuples and Patterns

We generalize a tuple to one in which each 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* 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 pattern p as:

$$\begin{aligned}\mathcal{M}(\theta, p) \equiv \langle \forall c : c \in p :: \langle \exists f : f \in \theta \wedge f.name = c.name \\ \wedge\ f.type\ \texttt{instanceof}\ c.type \\ :: c.constraint(f.value)\rangle\rangle.\end{aligned}$$ [3]

$\mathcal{M}$ requires that, for every constraint in the pattern, there is a field in the tuple with the same name, the same type (or a derived type), and a value that satisfies the constraint. 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.

[3] 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 $\forall$.

2.5 Basic Operations

Next, we classify the available operations, regardless of the tuple space structure. These operations fall into two categories: tuple generation and tuple retrieval. The former create new data items that agents can share for coordination purposes, while the latter allow agents to access available data items.

Tuple Generation. Agents create tuples using **out** operations: **out**(T, t), where T is a tuple space with a particular name located at a particular agent, and t is a tuple placed in T. In EgoSpaces, an **out** places the tuple in a local tuple space controlled by the generating agent. 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. For the purposes of access control, understanding tuple generation is important if agents can create tuples in other agents' tuple spaces. In these cases, the agent responsible for the target tuple space often desires the ability to express restrictions on the types of data that can be inserted or on which other agents can generate that data.

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 agent. Atomic probing operations, **rdp** and **inp**, guarantee, if a matching tuple exists, it is returned, but they can return ϵ if no match 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 these access operations entail only single tuples, many extensions 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 operation that returns all matching tuples from the tuple space. Access control for tuple retrieval operations is more obvious and natural than for the former tuple generation operations. The agent in control of the data items may desire some data to be read only, visible only to certain parties, or mutable only under certain conditions.

Different models present tuple space operations to the user in different ways. In LIME, agents operate over a federation of connected tuple spaces, while in EgoSpaces, agents operate over projections, called *views*, of all available data. These complex interactions can be reduced to the operations described above. We next investigate providing access control mechanisms for systems whose interactions can be expressed using this generalized tuple space model.

3 Access Control Function

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 control function 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.

3.1 Profiles

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.

3.2 Parameters

An access control function takes five parameters: the credentials, operation, tuple, pattern, and the owner's profile. We limit ourselves to these parameters because they capture the aspects of the coordination model we outlined previously. One could envision the inclusion of additional parameters that measure behaviors over the lifetime of the system, e.g., an access decision could be made based on the history of operations on a particular data item. We choose not to include those at this time because we feel the required bookkeeping overhead is not met by a demand from potential applications.

Credentials. Credentials allow an agent 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 may desire not to give away too much information, e.g., if the agent has multiple passwords, it should send only the correct one. However, this is not required in our access control mechanism because an agent's credentials are not directly exposed to other agents. These expressive credentials are especially beneficial in open and dynamic mobile environments, where it is often not possible to know a priori which agents can access restricted information. Instead, agents must prove they have required privileges. Agents select their credentials from the union of the host profile and the agent profile. The credentials are then presented as a tuple of attributes, which allows an 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. 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.

3.3 The Access Control Function Defined

Formally, the access control 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 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 request. The access decision cannot rely on arbitrary environmental properties, e.g., an agent cannot base a decision on the number of copies of a tuple. The access control function lends itself well to mobile environments because it allows adaptive policies. Access decisions are transparent to requesting agents; if access is denied, a requester does not even know that the matching tuple existed.

4 A Sample Implementation

The access control model is intentionally not presented in the context of any particular system. Instead, we have argued that it can be integrated with many tuple

space based coordination systems matching the form described in Section 2. As a demonstration of the feasibility and mechanics of such an integration, we have added this access control mechanism to a particular coordination middleware, EgoSpaces. We expect that, while some of the challenges we encountered are unique, other lessons learned will apply across coordination models.

In this section, we first highlight the novel features of EgoSpaces that make it amenable to coordination in ad hoc networks. This discussion also provides the information necessary to understand the integration of our access control mechanism. We complete this section with a technical description of the implementation of the access control mechanism within EgoSpaces. The description of the EgoSpaces model and middleware is intentionally brief. The interested reader can find a more careful evaluation of the model and its associated research concerns in the literature [6].

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 ad hoc 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 individual 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, a view's content automatically reflects the context changes without the agent's explicit intervention.

Practically, an agent defines its view as a set of constraints over the network, hosts, agents, and data. Within EgoSpaces each view is managed by an `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 communication between connected agents. The application agents implicitly use the `EgoManager` to define and interact with their views, which can require the `EgoManager` to interact with other `EgoManager`s (and, by association, other agents) on remote hosts. 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 operation request 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.

4.2 Integrating Access Controls with EgoSpaces

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 the tuples that qualify via their owning agent's access control function.

In providing access controls in EgoSpaces, we use credentials and access control functions along with the content-based retrieval and pattern matching mechanism already present in the system. Upon integrating the access control function, a set of credentials is now included as part of the view definition. These credentials are simply properties that convey information about the agent. The agent can alter its credentials at any time. To restrict other agents' perspectives according to their respective credentials, each agent also provides a dynamically modifiable access control function. A requesting agent's credentials are compared to the access control function of agents who contribute data to the view to restrict the tuples available in the view. With the access control functions in place, to evaluate a tuple for return to a requested operation an `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) providing the tuple and compares this information with the constraints defined in the requesting agent's view, *including the credentials*. The latter is the 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 requested operation can be performed.

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. 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 since access control may be expressed according to any property of the interacting agents.

The EgoSpaces system requires certain assumptions about its operating environment to provide atomic consistency guarantees regarding the performance of its operations on views. More details on these assumptions and dealing with environments where they do not hold can be found in [6]. Because the added access control provisions involve only local decisions at each contributing host, they have no negative impact on view consistency.

5 Programming with Access Control Mechanisms

In this section we demonstrate the use of access controls within the framework of the EgoSpaces coordination system. We first describe the programming interface

for using the access control function within EgoSpaces. We then describe two specific applications that use the described interfaces. We selected examples that apply in differing application domains to give a sense of the access control mechanism's flexibility. We do not give extensive details of the coordination mechanics specific to the EgoSpaces middleware but instead focus on the access control aspects of the two applications.

5.1 The Access Control API

Figure 1 shows the public API for defining and using credentials. As discussed in the previous section, an agent defines credentials that it sends with its view definition in EgoSpaces (or simple operations in other coordination systems) to identify itself to the other party. The first method in the Credentials interface, **selectProperty**, allows the agent to select a property from either the agent's own profile or its host profile to include in the credentials. The second method, **dropProperty**, allows an agent to remove a property from its credentials.

selectProperty(String profileType, String propertyName)
– select a property (identified by the propertyName) from either the host or agent profile (identified by profileType) to include in the credentials
dropProperty(String propertyName)
– drop the property identified by propertyName from the credentials

Fig. 1. The Credentials API Within the EgoSpaces Middleware

An agent who provides data defines an access control function to protect itself and its data. Each access control function is composed of one or more access control policies. The API for this component appears in Figure 2. The API contains three mechanisms to add a constraint to the access control policy. The first allows an agent to add a constraint that requires the credentials to contain a field with a certain name but no specific value. The second mechanism allows the agent to add a constraint that uses a built-in function (e.g., "=") to constrain the value of a named property in the credentials. The third and final mechanism allows the application agent to define a tailored constraint function that restricts the value of the named property in the credentials. This API also shows the method an agent uses to restrict the operations that can be performed on the coordination space. The final method evaluates the provided credentials to determine whether they match the constraints of this policy.

We provide the access control function as a disjunction of access control policies to allow more expressive functions. We require the combination of the credentials and the specific operation to satisfy at least one of the policies within the access control function. Figure 3 shows how agents assemble access control policies into a single access control function through an add method and a remove method. The **matches** method determines whether the credentials and the operation satisfy at least one of the access control policies.

addConstraint(String property)
– add a constraint that requires the existence of field with the given name
addConstraint(String property, String function, Serializable value)
– add a constraint that requires the named field to satisfy the given function and value
addConstraint(String property, ConstraintFunction cf)
– add a constraint that requires the named field to satisfy the given application-defined constraint function
addPermittedOperation(String operation)
– add the specified operation or operations to the list of those allowed
matches(Credentials)
– determines whether the provided credentials match this policy

Fig. 2. The AccessControlPolicy API Within the EgoSpaces Middleware

addPolicy(AccessControlPolicy acp)
– add the specified policy to the agent's access control function
removePolicy(AccessControlPolicy acp)
– remove the specified policy from the agent's access control function
matches(Credentials cred)
– determines whether or not the provided credentials match the policies contained within this access control function

Fig. 3. The AccessControlFunction API Within the EgoSpaces Middleware

5.2 A Music Sharing Application

A music sharing application for mobile users implemented on top of EgoSpaces serves as one vehicle for testing the access control implementation. The application provides users with access to a music service with sharing, search, and down-load capabilities. To determine what 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 4.

The data is also restricted according to the credentials 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. This phrase is provided as a product key when the user retrieves (purchases) the application from a reputable source (vendor). 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

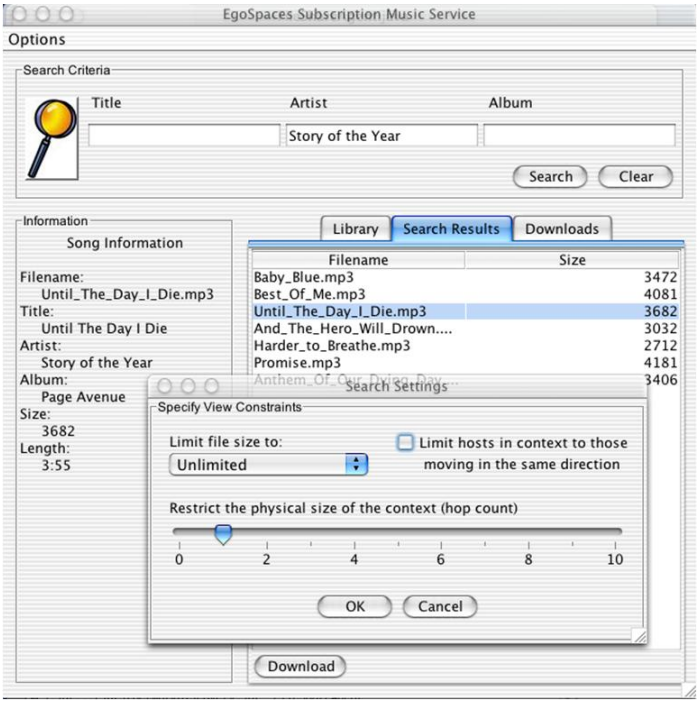

Fig. 4. The subscription music service

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();
c.selectProperty(AGENTPROFILE, ‘‘Passphrase’’);
```

First, the agent creates a credentials object. It then selects the passphrase property from the agent's profile that was handed out when the code was installed.

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

```
AccessControlPolicy policy = new AccessControlPolicy();
policy.addConstraint(‘‘Passphrase’’, ‘‘=’’, encryptedPhrase);
policy.addPermittedOperation(Operations.ALLRDS);
acf.addPolicy(policy);
```

In this code, the agent first creates a new policy. It then adds the single constraint that requires the passphrase to be equivalent to this agent's known encrypted phrase. It then adds to the permitted operations all read operations, preventing any admitted agents from removing any of the agent's own music files. Finally, the agent adds the defined policy to the access control function.

This music sharing application requires an initialization which can be arguably termed centralized. As indicated above, it can be equated with receiving

the software with a subscription, from a reputable source which provides the appropriate product key. After installing the music sharing software, users share music in a completely decentralized fashion, making autonomous decisions with no reliance on the availability of a centralized authority.

5.3 Administrative Domains

Many applications restrict agent operations to administrative domains. Assume nested domains defined as a university, a department, and a research group. 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 perform an **in** operation for a license from a connected tuple space. If successful, the number of available licenses decreases by one. When the user finishes using the software, the agent replaces the license in its local tuple space.

To define the credentials in this application, an agent requesting a license uses the following code:

```
Credentials c = new Credentials();
c.selectProperty(HOSTPROFILE, ‘‘University’’);
c.selectProperty(HOSTPROFILE, ‘‘Department’’);
c.selectProperty(HOSTPROFILE, ‘‘Group’’);
```

The agent creates an empty credentials object and then selects three properties from the host's profile to add to the credentials: the university, the department, and the group. These three characteristics will be used to determine whether the agent has the right to access the license it requests.

Agents responsible for the licenses protect them by using access control functions that restrict access based on the administrative domains outlined above. This access control function is defined using the following code:

```
AccessControlPolicy policy = new AccessControlPolicy();
policy.addConstraint(‘‘University’’, ‘‘=’’, ‘‘WUSTL’’)
  .addConstraint(‘‘Department’’, ‘‘=’’, ‘‘CSE’’)
  .addConstraint(‘‘Group’’, ‘‘=’’, ‘‘mobi’’);
policy.addPermittedOperation(Operation.SINGLES);
acf.addPolicy(policy);
```

After creating a new policy, the agent adds three constraints to it that restrict the university, department, and group to the correct set of users. The agent

permits all single operations (that interact with only a single license at a time). Finally, the policy is added to the access control function.

As these two examples demonstrate, the developer burden for adding access control to the application is minimized and builds on the notion of tuples and tuple spaces to ease the learning curve for the application programmer.

6 Discussion

The access control function provides a flexible mechanism for specification of dynamic and adaptive privileges in mobile systems. Next, we take a deeper look at two aspects of the access control function: expressiveness and overhead.

6.1 Expressiveness

While its expressiveness makes the access control function flexible and useful in coordination among constantly changing mobile agents, this flexibility comes at some cost. On one hand, because credentials can encode arbitrary information about an agent, particular applications can adapt credentials to their needs. In addition, because the access control function takes a number of parameters, an agent can dynamically adjust its policies. However, while complex policies are possible, constructing the function (from the developer's perspective) can become difficult as policies become more complex. Fortunately, because the design employs the use of pattern matching, much of this complexity can be hidden by the infrastructure.

6.2 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 communication overhead.

Additional Computation. Because the function can contain arbitrary code, its 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.

7 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 [18] 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 [19],, tuples are unordered sequences of individually encrypted fields, and, to match an encrypted field, a pattern must contain a correct key. Other work [20] 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) [21] 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. One aspect of LGI that separates it from the access control mechanism described in this paper is that it allows access rules to be imposed from outside the individual agents. We do not consider such cases in our work because it departs from our view that agents should be as autonomous as possible.

The Smart Messages system [22] structures a mobile computing system in much the same way as discussed in this paper. Using Smart Messages, however, the coordination in the system occurs through the logical migration of Smart Messages. In this system, access control takes the form of admission control in determining when to allow migrating Smart Messages to execute on a new host. The admission managers responsible for this task use information about the resource needs of an arriving Smart Message as they relate to the available resources on the node. The access control mechanism described in this paper can account for more varied information than resource availability by using credentials describing the application agent and using the data items themselves when making access decisions.

Work targeted directly to ad hoc networks [23] begins to address the need for credential verification among interacting parties using X.509 certificates. This work focuses on adapting the chain of verification for certificates to function in an ad hoc network by using assertions generated by peers in the ad hoc network. The disadvantage of applying this type of solution in the environments we have described is that it requires some a priori knowledge shared among the peers in the ad hoc network in order to be able to verify the credentials of other participants. Key pre-distribution schemes targeted to sensor networks [24] have worked without a centralized server to establish pairwise secure communications. These approaches generally focus on maximizing the total security of the system to successfully handle more "compromised" nodes. These schemes focus simply on providing the ability to encrypt data and do not address the need to restrict access to certain data items based on contextual properties.

Additional work on authentication protocols in ad hoc networks [25, 26] focuses on securing communications among parties in ad hoc networks. These protocols tend to attempt to validate the identity of a communicating party. Our work instead focuses on the data sharing aspects and assumes that agents do not necessarily care about the exact identity of a coordinating partner, but about properties of the partner. This style of access control is more in line with our target environment since we assume that an application does not have a priori knowledge of the other agents or data it will interact with. The flexible nature of the access control mechanism described in this paper allows agents to base access decisions on abstract properties and the content of data, enabling more expressive access rules.

8 Conclusion

In today's emerging mobile systems, applications find themselves structured as networks of mobile agents that must interact to achieve the users' goals. As mobile devices become increasingly prevalent and more users join mobile networks, the complexity of mediating interactions among agents multiplies. A significant roadblock to the widespread deployment of many mobile applications lies in the inability to secure interactions in this open environment where encounters with others are necessarily opportunistic and unpredictable. The work presented

in this paper examined one aspect of this need by introducing a mechanism for agents to control access to data. This mechanism, in the form of an agent-tunable function, allows autonomously operating agents to share data with other connected agents, given some restrictions. Each agent makes individual access decisions for the data item it "owns" based on numerous properties including properties of the environment, of the agent's state, of the requesting agent, and even properties of the data item itself. By placing control in the hands of individual agents, we have eliminated the need for a centralized authority to make access decisions and thus created an access mechanism that functions in ad hoc networks where the coordinating parties are not known in advance. Because each access control decision is independent and made in a decentralized manner, the access control function naturally scales to networks of high numbers of mobile agents. Because we started with a foundational model of coordination, the resulting mechanism addresses the access control needs within mobile coordination models. In particular, the construct provides increased scalability and decoupling when compared with previous constructs 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. Gelernter, D.: Generative communication in Linda. ACM Transactions on Programming Languages and Systems **7** (1985) 80–112
2. Kaufman, J., Lehman, T.: OptimalGrid: The almaden SmartGrid project: Autonomous optimization of distributed computing on the grid. IEEE Task Force on Cluster Computing **4** (2003)
3. Wyckoff, P., McLaughry, S., Lehman, T., Ford, D.: TSpaces. IBM Systems Journal **37** (1998)
4. Freeman, E., Hupfer, S., Arnold, K.: JavaSpaces Principles, Patterns, and Practice. Addison-Wesley (1999)
5. Murphy, A.L., Picco, G.P., Roman, G.C.: LIME: A middleware for physical and logical mobility. In: Proceedings of the 21^{st} International Conference on Distributed Computing Systems. (2001) 524–533
6. Julien, C., Roman, G.C.: Egocentric context-aware programming in ad hoc mobile environments. In: Proceedings of the 10^{th} International Symposium on the Foundations of Software Engineering. (2002)
7. Cabri, G., Leonardi, L., Zambonelli, F.: MARS: A programmable coordination architecture for mobile agents. Internet Computing **4** (2000) 26–35
8. Moore, J.: Mobile code security techniques. Technical Report MIS-CIS-98-28, University of Pennsylvania (1998)

9. White, J.: Telescript technology: The foundation for the electronic marketplace. General Magic White Paper, General Magic, Inc. (1994)
10. Gray, R., Kotz, D., Cybenko, G., Rus, D.: D'Agents: Security in a multiple-language, mobile-agent system. In Vigna, G., ed.: Mobile Agents and Security. Volume 1419 of LNCS. Springer-Verlag (1998) 154–187
11. Gray, R.: Agent tcl: A flexible and secure mobile-agent system. In: Proceedings of the 4^{th} Annual Tcl/Tk Workshop. (1996)
12. Necula, G.: Proof-carrying code. In: Proceedings of the Symposium on Principles of Programming Languages. (1997)
13. Sander, T., Tschudin, C.: Protecting mobile agents against malicious hosts. In Vigna, G., ed.: Mobile Agents and Security. Volume 1419 of Lecture Notes in Computer Science. Springer-Verlag (1998) 44–60
14. Borselius, N., Mitchell, C.J., Wilson, A.: Undetachable threshold signatures. In: Cryptography and Coding—Proceedings of the 8^{th} IMA International Conferemce. Volume 2360 of LNCS. (2001) 239–244
15. Papadimitratos, P., Haas, Z.: Secure data transmission in mobile ad hoc networks. In: Proceedings of the 2003 ACM Workshop on Wireless Security. (2003) 41–50
16. National Center for Supercomputing Applications, Integrated Decision Technologies Group: SAMCat: A securable active metadata catalogue. (2002)
17. Byrd, G., Gong, F., Sargor, C., Smith, T.: Yalta: A secure collaborative space for dynamic coalitions. In: IEEE 2^{nd} SMC Information Assurance Workshop. (2001)
18. Cremonini, M., Omicini, A., Zambonelli, F.: Coordination and access control in open distributed agent systems: the TuCSoN approach. In Porto, A., Roman, G.C., eds.: Coordination Languages and Models. Volume 1906 of LNCS., Springer-Verlag (2000) 99–114
19. Bryce, C., Oriol, M., Vitek, J.: A coordination model for agents based on secure spaces. In Ciancarini, P., Wolf, A., eds.: Proceedings of the 3^{rd} International Conference on Coordination Models and Languages, Springer-Verlag (1999) 4–20
20. Handorean, R., Roman, G.C.: Secure servise provision in ad hoc networks. In: Proceedings of the 1^{st} International Conference on Service Oriented Computing. (2003)
21. Minsky, N., Minsky, Y., Ungureanu, V.: Safe tuplespace-based coordination in multi agent systems. Journal of Applied Artificial Intelligence **15** (2001)
22. Kang, P., Borcea, C., Xu, G., Saxena, A., Kremer, U., Iftode, L.: Smart messages: A distributed computing platform for networks of embedded systems. The Computer Journal Special Issue on Mobile and Pervasive Computing ((to appear))
23. Keoh, S.L., Lupu, E.: Towards flexible credential verification in mobile ad hoc networks. In: Proceedings of the ACM Workshop on Principles of Mobile Computing. (2002) 58–65
24. Du, W., Deng, J., Han, Y.S., Varshney, P.K.: A pairwise key pre-distribution scheme for wireless sensor networks. In: Proceedings of the 10^{th} ACM Conference on Computer and Communication Security. (2003) 42–51
25. Weimerskirch, A., Thonet, G.: A distributed light-weight authentication model for ad hoc networks. In: Proceedings of the 4^{th} International Conference on Information Security and Cryptology. (2001) 341–354
26. Balfanz, D., Smetters, D.K., Stewart, P., Wong, H.C.: Talking to strangers: Authentication in ad hoc wireless networks. In: Network and Distributed System Security Symposium. (2002)