Report Number: WUCSE-2003-34

2003-04-29

Managing Access Control in the Presence of Physical and Logical Mobility

Christine Julien, Gruia-Catalin Roman, and Jamie Payton

The emerging mobile computing environment draws new attention to the need for co-ordination among networked components. The very nature of this environment requires parties to interact even when they have never met before, and subsequent encounters are totally unpredictable. Because mobile networks are often decoupled from any fixed network infrastructure, reliance on centralized servers to authenticate agents and to establish data access policies is impractical. Access control is a key component of security in such systems, and application agents must be able to directly manipulate and examine policies because they need full and flexible control over their data. Starting... **Read complete abstract on page 2.**

Managing Access Control in the Presence of Physical and Logical Mobility

Christine Julien, Gruia-Catalin Roman, and Jamie Payton

Complete Abstract:

The emerging mobile computing environment draws new attention to the need for co-ordination among networked components. The very nature of this environment requires parties to interact even when they have never met before, and subsequent encounters are totally unpredictable. Because mobile networks are often decoupled from any fixed network infrastructure, reliance on centralized servers to authenticate agents and to establish data access policies is impractical. Access control is a key component of security in such systems, and application agents must be able to directly manipulate and examine policies because they need full and flexible control over their data. Starting from this premise, we examine the essential features of general access control policies designed to respond to the specific needs of agent coordination in the presence of logical and physical mobility. A novel construct to support such policies is proposed and evaluated with respect to its impact on mobile applications. We also show some example uses of this access control construct.

2003-34

Managing Access Control in the Presence of Physical and Logical Mobility **PLEASE SEE WUCSE-03-70**

Authors: Julien, Christine; Roman, Gruia-Catalin; Payton, Jamie

April 29, 2003

Abstract: The emerging mobile computing environment draws new attention to the need for coordination among networked components. The very nature of this environment requires parties to interact even when they have never met before, and subsequent encounters are totally unpredictable. Because mobile networks are often decoupled from any fixed network infrastructure, reliance on centralized servers to authenticate agents and to establish data access policies is impractical. Access control is a key component of security in such systems, and application agents must be able to directly manipulate and examine policies because they need full and flexible control over their data. Starting from this premise, we examine the essential features of general access control policies designed to respond to the specific needs of agent coordination in the presence of logical and physical mobility. A novel construct to support such policies is proposed and evaluated with respect to its impact on mobile applications. We also show some example uses of this access control construct.

Managing Access Control in the Presence of Physical and Logical Mobility

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

Mobile Computing Laboratory
Department of Computer Science and Engineering
Washington University
Saint Louis, MO, USA

Abstract

The emerging mobile computing environment draws new attention to the need for coordination among networked components. The very nature of this environment requires parties to interact even when they have never met before, and subsequent encounters are totally unpredictable. Because mobile networks are often decoupled from any fixed network infrastructure, reliance on centralized servers to authenticate agents and to establish data access policies is impractical. Access control is a key component of security in such systems, and application agents must be able to directly manipulate and examine policies because they need full and flexible control over their data. Starting from this premise, we examine the essential features of general access control policies designed to respond to the specific needs of agent coordination in the presence of logical and physical mobility. A novel construct to support such policies is proposed and evaluated with respect to its impact on mobile applications. We also show some example uses of this access control construct.

1 Introduction

In recent years, computing devices have become ubiquitous. These devices often possess the ability to communicate wirelessly with other devices, opportunistically forming wireless networks not necessarily connected to a wired infrastructure. In such environments, distributed application components need to communicate to exchange information and services or to coordinate tasks. These networks can include a handful of devices or thousands of heterogeneous components, making coordinating and mediating their competing needs a massive task. Much work has been extended on developing practical coordination models to facilitate rapid application development for this demanding application domain.

This paper focuses on a broad class of coordination models that utilize tuple spaces as the basis for decoupled coordination. The Linda model [7] provides a centralized tuple

[1] Email: {julien, roman, payton}@cse.wustl.edu

space where application agents deposit and retrieve information using content-based matching of patterns against data items, thereby coordinating with other agents without ever having to encounter them. Variations on this theme adapt Linda to the dynamic mobile environment where coordination through a central repository is not feasible. Due to the open and dynamic nature of these systems, security concerns of three types arise: the need to protect mobile hosts from malicious agents, the need to protect agents from tampering hosts, and the need to protect data. The D'Agents system [8] uses public-key cryptography to authenticate incoming agents to increase the security of hosts. The more difficult problem of protecting agents from hosts has received increased attention of late. Undetachable threshold signatures [1], prevent hosts from tampering with an agent's data and also distribute authority among multiple agents.

Mechanisms for protecting data can be divided into two categories: ensuring data integrity and controlling data access. Much research has focused on encrypting communication within coordination spaces. SAMCat [13] uses SSL encryption and authentication to securely transmit tuples into and out of a globally persistent data space. Yalta [3] encrypts communication to and from the tuple space, requiring users attempting to access the data to first be authenticated. 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, permanent disconnection from a wired infrastructure is not uncommon, rendering a centralized solution impossible.

A common mechanism for addressing access control concerns uses access matrices to describe the rights that subjects have regarding objects. The rows of the matrix correspond to users, and the columns correspond to objects. Cells in the matrix contain the access rights that user has on a particular object. The access matrix approach is a generalization of several commonly used approaches, including access control lists and capability definitions.

Given our coordination model's expressiveness and the mobile environment in which it operates, applying an access matrix to achieve access control policies is not straightforward. First, the manner of data access should include not only the issued operation but also the pattern used to select the data. In some cases, the components of this pattern provide information about an application's prior knowledge of the data. The owning agent may want to restrict access to agents who know the "correct" way to access the data (providing a wild card pattern that matches any tuple allows an agent to access any piece of data and gives the requesting agent a lot of power). These observations led us to investigate a more general access control mechanism for mobile coordination systems.

This is in sharp contrast with the use of an access control matrix which requires knowing all of the coordinating parties in advance. The number of possible agents, combined with the amount of data available over the lifetime of the system, generates a very large matrix that is likely to be sparsely populated. The access control function introduced in this paper overcomes the limitations of applying an access matrix based approach in mobile systems by operating over general descriptions of the coordinating parties and dynamically adjusting to the changing context.

Before we discuss the specifics of the access control function, we introduce a generalized coordination model for mobile systems. Section 3 describes the access control function, provides details of its components, and shows some examples. In Section 4, we discuss the construct's expressive power and the overhead incurred in using it. Section 5 overviews related work, and conclusions appear in Section 6.

2 A Generalized Coordination Model

In this section, capture the essential features of tuple space based coordination models in an attempt to explain the access control requirements for mobile coordination middleware. The result is a generalization that spans the gamut from tuple definition to sophisticated operations. To accomplish this, we focus on Linda's content-based style of coordination. In the original Linda model, processes generating tuples in a centralized tuple space (via an **out** operation), and processes interact with the producers via **in** operations (to remove tuples), and **rd** operations (to simply read tuples). Both of these operations execute in a content-based manner in which the retrieving operation 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 Linda operations decouple agents in a manner useful in mobile networks, as demonstrated below.

2.1 Data Representation: The Tuple Space

We assume a class of coordination models based on tuple spaces, by and large the dominant paradigm in coordination literature. Some systems (e.g., MARS [4] and TuCSoN[14]) focus specifically on logically mobile agents moving across a network of physically stationary hosts. Other systems (e.g., Lime [12] and EgoSpaces [10]) address the integration of physical and logical mobility. In all such systems, a Linda tuple space is associated with a component of the network that maintains the availability of the data and allows other agents in the system access to the data. In some systems, these tuple spaces are permanently bound to hosts, in other cases they are bound to agents, and in still other cases the tuple space is distributed throughout the network among some combination of hosts and agents.

Fundamental to all these models for coordination in mobile environments is the use of a Linda-like tuple space. The actual distribution of the tuples in the system is irrelevant with respect to the access control mechanisms described in this paper. The key aspect of the representation is the manner in which application agents interact with the data space. Our model assumes a tuple space bound to each mobile agent. Using this model, we can simulate other models if necessary, e.g., to simulate tuple spaces bound to a host, we associate an agent permanently to a host and use its tuple space to represent host's tuple space. Next, we describe an enhanced and more flexible definition of the tuple construct, one that subsumes and unifies all existing perspectives visible in models to date.

2.2 Tuples, Patterns, and the Matching Function

Original Linda tuples and patterns contain a sequence of fields. Each field is of a particular type that can be either an actual or a formal; an actual contains a value of the field's type, while a formal is a place holder for the field's type. Agents' operations select tuples by matching the tuple against a pattern. A pattern matches a tuple if they have the same number of fields and every field in the pattern matches the corresponding field in the tuple. Two fields match if they contain the same actual value, they contain the same formal, or the pattern's field contains a formal of the same type as the tuple field's value.

The Linda extensions discussed above use similar notions of tuples and patterns. Our model generalizes the Linda tuple representation to one in which the fields are identified by a name, allowing patterns and tuples of different lengths. A tuple is an unordered set of triples of the form: $\langle (name, type, value), (name, type, value), \ldots\rangle$. For each field, *name* is the name of the field, and *type* is the data type of *value*. In a given tuple, the *name* of each field must be unique. Fundamentally, users access tuple spaces by matching patterns against contents of tuples. While adhering to the content-based nature of Linda pattern matching, we extend the traditional semantics to allow the provision of more flexible constraints over fields. A pattern takes the form: $\langle (name, type, constraint), (name, type, constraint), \ldots\rangle$. The *constraint*s are functions that provide requirements that the *value* in a field must match for the field in the tuple to match the field in the pattern. More specifically, the matching function $\mathcal{M}$ is defined over a tuple θ and a pattern p as:

$$\begin{aligned}\mathcal{M}(\theta, p) \equiv \langle \forall c : c \in p ::&\\ &\langle \exists f : f \in \theta \wedge f.name = c.name \wedge f.type \texttt{ instanceof } c.type\\ &\quad :: c.constraint(f.value)\rangle\rangle.\end{aligned}$$ [2]

$\mathcal{M}$ requires that, for every constraint c in the pattern, there must be a field f 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 contained in the pattern but can contain additional fields.

2.3 Basic Operations

This section classifies the operations available in mobile coordination systems, regardless of the structure of the tuple space. Basic operations can be divided into two groups: tuple generation operations and on-demand access operations

Tuple Generation. Coordination systems based on Linda provide a tuple generation mechanism; we retain Linda's name for this operation: **out**. While variations exist, the essence of tuple generation is that a tuple may be placed in a specific tuple space: **out**(T, t). This operation places the tuple t in the tuple space T, i.e., 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. In LIME, this operation can place a tuple in any tuple space owned by any agent on a connected host. In MARS the generated tuple is created in the tuple space associated with the local host. In any case, the agent responsible for the tuple space will want some control over what tuples can be inserted in its tuple space.

Tuple Retrieval. Mobile coordination systems, in general, supply two types of operations derived directly from Linda's **rd** and **in** operations. In the mobile systems, these two types of operations each assume three basic forms: blocking, atomic probing, and scattered probing. We retain operations with the strong original semantics because even mobile applications sometimes require strong guarantees. The standard form for these operations is **rd**(T, p), which returns a tuple matching the provided pattern p

[2] The three-part notation $\langle \textbf{op}\ quantified\_variable : range :: expression\rangle$ is defined as follows: The variables from *quantified_variables* take on all possible values permitted by *range*. Each such instantiation of the variables is substituted in *expression*, 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 three-part expression is the identity element for **op**, e.g., *true* when **op** is $\forall$ or zero if **op** is "+" .

from the tuple space T. As with tuple generation, the tuple space can be either local to the agent or controlled by some other party in the network. The second two forms of operations, atomic probing and scattered probing, stem from the observation that features of some mobile applications can make synchronous semantics unwieldy. Atomic probing operations guarantee that if a matching tuple exists in the specified tuple space, it is returned, but the operations do not block until a matching tuple exists. We will refer to these operations as **rdp** and **inp**. Like the synchronous Linda operations, these operations are atomic with respect to the tuple space on which they are issued, and in some cases in the mobile environment, guaranteeing this atomicity can be difficult and expensive. For these cases, our coordination model also provides scattered probing operations that offer even weaker guarantees: **rdsp** and **insp**. While retrieval operations most often entail only single tuples, constructs exist in mobile coordination models that allow simultaneous access to groups of tuples. These operations come in all three forms described in the previous paragraph 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.

Some of the models (e.g., LIME and EgoSpaces) present these tuple space operations to the user in a different manner. In LIME, application agents actually operate over a federation of connected tuple spaces, while in EgoSpaces agents operate over projections of all available data, called *views*. In these cases, however, these more complex interactions can be reduced to either the tuple space operations here or the wide-spectrum operations discussed next.

2.4 Wide-Spectrum Operations

Over time, some coordination models have defined new operations that do not fit the paradigms described above but further facilitate coordination in the dynamic mobile environment. The earliest of these additions, seen in LIME and MARS, were reactive constructs. This initial development of new operations was motivated by the observation that if an agent needs to wait for a particular piece of data before performing additional actions it must either block or poll. These costly and inefficient mechanisms prevent the agent from performing other work in the meantime. The reactive constructs allow agents to register interest in particular pieces of data. When data matching the registration appears in the tuple space, code specified by the agent is executed. The need to react to events in the tuple space also appeared as coordination systems matured. For example, an agent may want to be notified if another agent accesses a particular piece of data.

Our coordination model encompasses these behaviors. In our experience in facilitating application development in mobile environments, we see that ostensibly different applications use very similar behaviors. Our model introduces a behavioral construct that provides, as an integral part of the coordination model, these commonly used behaviors in an extensible and flexible fashion. These behaviors include the ability to transparently migrate certain data from one tuple space to another and the ability to transparently duplicate data encountered in the environment.

3 Access Control Function

Our coordination model provides flexible and efficient communication in the face of constantly moving hosts, agents, and data. In this environment, security concerns become of paramount importance. In our model, an agent assumes responsibility for mediating access to tuples stored in its local tuple space. The ability to control access in this manner is fundamental to coordination systems targeted to the mobile environment because it allows the access control policies to reflect the changing needs of an agent. To accomplish this each agent specifies an individualized access control function that limits the ability of other agents to access its local data.

Our novel access control model utilizes tuple space based coordination to control access to data at a fine granularity. An agent responsible for a particular data item can restrict which agents access its data and the manner in which the access occurs. To accomplish the former, the owning agent requires a requesting agent to identify itself via a set of credentials. To accomplish the latter, an access control function can account for the particular access operation when deciding whether or not to allow access. In the end, each agent defines a single access control function that takes as parameters a particular tuple, a set of credentials identifying the requesting agent, the specific operation being performed, the pattern used to access the data, and the profile of the owning agent. This function returns a boolean value indicating whether or not the requested access should be allowed.

3.1 Profiles

Before describing the access control function in more detail, we introduce the profiles that maintain properties of hosts and agents. While these can be viewed as two separate profiles, we will assume that an agent's profile logically contains the profile of the host on which it is running. This profile provides an abstraction of contextual information regarding the particular agent and host. We can represent this profile as a tuple, allowing powerful content-based operations on the contextual data. Particular applications or coordination systems may require specific attributes in this profile. In general, we always assume that a profile contains at least a unique host ID differentiating the host from all other hosts and a unique agent ID.

3.2 Access Control Function Parameters

An access control function takes five parameters: the tuple, the requesting agent's credentials, the operation, the pattern used in the request, and the owner's profile.

Credentials. Credentials are the only mechanism by which an agent can convey information about itself to other agents. Our access control model allows an agent to determine access based on arbitrary properties of the requesting agent. Upon requesting operations, an agent sends a portion of its profile as credentials that identify it. In the simplest cases, the information sent in the credentials can be a standard set of information, e.g., the agent's ID, a third-party authentication, etc. In other cases, when the requesting agent has some a priori knowledge about the access requirements, these credentials can be more complicated, e.g., a password. When constructing credentials, the agent must be careful not to give information away to other agents with which it is coordinating, e.g., if the agent has multiple passwords for different uses, it should send only the correct one.

The need for this form of identification from requesting agents is of particular importance in mobile applications, which are open and dynamic systems in which coordinating agents appear and disappear sporadically. In these applications, it is often not possible to know a priori exactly which agents can access restricted information. Instead, these agents need to prove they have the needed privileges through other means. The credentials provide exactly this alternative.

A set of credentials is a subset of the agent's profile that can be viewed as a tuple of attributes: $\langle(att\_name, type, value), (att\_name, type, value), \ldots\rangle$. The credentials and their transmission with the operation request are assumed to be private. This layer of security is outside the scope of this paper but could be accomplished using any number of cryptography schemes already under development. Because the credentials are a tuple, the access control function can use the pattern-matching scheme inherent in the coordination system to evaluate credentials.

Operation. The access control function can also account for the exact operation used to access the data. For example, it is often the case that some data should be restricted to read-only access. Current mobile coordination systems do not inherently provide this mechanism. This restriction based on the particular operation coupled with the provision of credentials allows an agent to decide on a per-agent, per-operation basis whether or not to grant access to a particular tuple. This allows dynamic applications to allow one set of operations for some agents, but perhaps a more restricted set of operations for others.

The possible operations are represented as a set O which contains string representations of the different operations (e.g., "rd", "reaction", "migration"). When evaluating the access control function, a single element of O that identifies the operation is passed as a parameter to the access control function.

Requested Tuple. The key to providing a fine granularity of control is allowing agents to determine access privileges based on individual data items. Because we are focusing on tuple space based coordination models, this can be accomplished by allowing the access control function to operate over the tuple to be returned from by an operation. The pattern-matching capabilities of our model allow this portion of the access control function to be easily defined while remaining sufficiently flexible.

Pattern. A powerful component of the access control function is its ability to determine privileges based on the pattern used in the content-based operation. As discussed previously, the pattern gives the owning agent some information about the requesting agent's knowledge of the data is attempting to access. Some knowledge of the structure of the requested tuple might indicate that the requesting agent shares common application goals, giving an implicit indication of trust.

Owner's Profile. The final parameter of the access control function considers the owner's current state. Because the access policy is determined dynamically, access can be granted based on contextual information. For instance, 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 Access Control Function

We now focus on the access control function's definition. This function takes the five parameters described above, and determines whether or not to allow the requested access. Formally, this function can be represented as:

$$\mathrm{ACF} : T \times C \times O \times P \times \Pi \rightarrow \{0, 1\}$$

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:

$$access = \mathrm{ACF}(credentials_r, operation, tuple, pattern, profile_o)$$

where r is the agent requesting the operation and o is the tuple's owner.

We will show the expressive power of this construct later in this section. For now we consider what this access control function *cannot* naturally or easily represent. First, access decisions cannot be based on properties of the requesting agent not included in its credentials. The decision cannot be based on arbitrary properties of the environment that are not part of the owning agent's profile. For example, an agent cannot base a decision on the number of copies of a tuple in the tuple space because that information is not part of the access control function's components. This kind of behavior might be desired if destructive reads are allowed unless they attempt to remove the last copy of a tuple.

The access control function lends itself well to the dynamic mobile environment because it allows access policies to adapt to the context of coordinating agents. The access decision is transparent to the requesting agent; if access is denied, the requester does not even know that the matching tuple existed.

3.4 Using the Access Control Function

In this section, we show how real-world coordination systems use access control functions. We first show how the access control function benefits a particular coordination system, EgoSpaces. We then return to our generalized coordination model and show how restricting operations to administrative domains can be implemented with the construct we have presented.

Use in EgoSpaces Model. One of the first coordination models to place access control in a central position was EgoSpaces, which addresses the needs of agents operating in large-scale heterogeneous environments. In this system, an agent operates over a context that can include, in principle, all data available in the entire network. EgoSpaces' unique model of coordination, however, is based on the observation that providing access to this vast amount of information can prove costly. Therefore, EgoSpaces structures data in terms of *views*, or projections of the maximal set of data. Each agent defines its own views specialized to it's needs; these individualized views provide a much needed abstraction of the dynamic environment by constraining properties of the network, hosts, agents, and data. To further reduce programming costs, EgoSpaces transparently maintains these views; as hosts and agents move, the view's contents automatically reflect contextual changes without the agent's explicit intervention.

Due to the nature of view definitions, EgoSpaces employs the agent-specified access control function on a per-view basis. When a reference agent defines a view, it attaches a set of credentials and a list of operations it intends to perform over the view. The EgoSpaces system uses each potential contributing agent's access control function to determine which tuples belong in the view. The view membership is still determined on a per-tuple basis. In the end, the view contains only tuples that qualify via their owning agent's access control function.

Administrative Domains. Another class of applications that can benefit from the generalized access control function involves the restriction of agent operations to administrative domains. Because all components in our system are mobile, the domains themselves might also be mobile. For our example, let's assume the domains are defined as a university's computer system (identified via a host id's network prefix), a departmental computer system, and a research group's computer system. To provide security guarantees, certain applications may want to limit access to certain pieces of data to only computers on the university's network. Still other data ought to be restricted to departmental computers, or to research group computers.

As one example, consider a user in the research group who wants to use a software license of which the research group has n copies. Given our coordination model, the licenses are stored as tuples in some tuple space, which can be viewed as a distributed service repository. We assume all of the members of the research group work on physically mobile hosts. Each host can carry its own repository, the licenses are initially distributed in some random fashion, and another user in the group can take control of a license in another's repository if the computers are within communication range. The agents controlling these repositories restrict access to requests from another agent whose user is a member of the group, who has a departmental authentication (retrieved a priori), and is running on a computer in the university domain. To retrieve a license, a user provides these three things as credentials and attempts to **in** a license from its local repository or a connected one. If successful, the number of available licenses decreases by one. When the user finishes using the software, it replaces the license in its local repository.

4 Analysis and Discussion

The access control function provides a flexible mechanism for agents to specify privileges dynamically and adaptively in mobile coordination models. This work relies on encryption mechanisms that lie outside the scope of this paper to protect the message transmissions. We assume, however, that this can be accomplished through the use of an encryption scheme.

4.1 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, the fact that credentials can encode arbitrary information about an agent makes them extremely expressive and allows particular applications to adapt credentials to their needs. This may introduce communication overhead. On the other hand, a requesting agent needs to take care not to reveal too much information. Any information sent in credentials is no longer completely secret. If the requesting agent knows five passwords and sends them with every request, any other agent that receives its credentials knows those passwords.

Functions. Because the access control function takes as parameters a number of facets of the coordination, it can dynamically adjust a particular agent's access policies. Again, flexibility comes with a cost. While it is possible to design complex access control policies, 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 the complex policies.

4.2 Overhead

Given the model's expressiveness, it is useful to evaluate the overhead of the new mechanism. First, the addition of the access control mechanism we describe introduces some amount of programming overhead. This overhead is difficult to quantify without a wide spread case study involving actual users implementing actual access control policies. While this is a useful future task, it is outside the scope of this paper. Also useful, however, is some measure of the additional communication required to exchange the necessary information and the additional computation needed to evaluate the function.

Additional Communication. In evaluating the communication overhead. we will quantify the communication costs in terms of the time required for a transmission between two connected parties, $t_{a,b}$. We refer to the transmission rate between hosts a and b as $rate_{a,b}$. Before adding access control mechanisms, the time required to send a message from host a to host b is:

$$t_{a,b} = \frac{(|op| + |pattern| + |agent\_id_r|)}{rate_{a,b}}$$

$|op|$ is the number of bits required to identify the operation; this will be some constant across successive operations, dependent upon the number of possible operations. $|pattern|$ is the number of bits required to represent the pattern. This 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.

We can write a similar term to express the amount of time a transmission takes when using the access control function:

$$t^{acf}_{a,b} = \frac{(|op| + |pattern| + |host\_id_r| + |credentials_r|)}{rate_{a,b}} = t_{a,b} + \frac{|credentials_r|}{rate_{a,b}}$$

The added time is due to the need to transmit credentials to gain access.

One way to express the communication overhead is through the ratio of the transmission time with credentials to the transmission time without credentials:

$$overhead_{a,b} = \frac{t_{a,b} + \frac{|credentials_r|}{rate_{a,b}}}{t_{a,b}} = 1 + \frac{|credentials_r|}{rate_{a,b} * t_{a,b}}$$

Credentials are essentially a tuple. An interesting observation gleaned from the above relationship is that if we assume that the number of bits required to completely represent a pattern is greater than that required to represent either the operation or the identity of the agent, the size of the pattern dominates the transmission time. This assumption is likely to hold true because the operation and agent identity are simple data types, while a tuple is a much more complex structure. In this case, if the credentials are approximately equivalent in size to the pattern, then the overhead of using access control is approximately equal to 2.

An application has direct control over the amount of overhead it incurs because it individually 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. Evaluation of the access control function does require some additional computation on the host where the data being accessed is located. From the coordination system's perspective, implementation of the access control function requires a single method invocation into the agent's code. However, 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 coordination system can include a mechanism to allow a host to prevent undesirable access control functions by bounding the time they are allowed to run or by imposing restrictions on their capabilities.

5 Related Work

As discussed in Section 1, the access matrix approach does not directly lend itself to mobile coordination systems. One example of attempting to apply such a method is demonstrated in [5], in which TuCSoN agents are assigned capabilities defining tuple space operations for particular patterns in a certain tuple space. These capabilities are stored in an access control list (ACL) for that tuple space. Such an approach requires that all coordinating parties are known in advance and that some centralized party determines access policies statically. The role based access control (RBAC) model [6] replaces this knowledge by roles agents assume, where each role is associated with a set of allowed operations on a set of objects. An agent's roles can change over time, so the set of operations an agent can perform is dynamic. An agent's role in the access matrix is evaluated each time it performs an operation on an object. This again requires knowledge of all of the possible roles, and assumes a central authority to distribute roles.

Other systems use encryption schemes for access control. In SecOS [2], tuples are unordered sequences of individually encrypted fields. Tuple matching is extended so that, to match an encrypted field, a pattern must contain a correct key. In LimeGuard [9], keys are associated with tuple spaces, and an agent must provide the key to access the tuple space. This system also provides a variety of tuple level access control capabilities via passwords. 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) [11] provides an expressive context-sensitive approach to access control without encryption. In LGI, a group of agents that wish to interact via a tuple space 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 the arrival or departure of tuple space operations. Programming applications in LGI requires programming specific actions in the access control policy and adding a controller to each agent to mediate tuple space requests. In contrast, in our model, programming takes place in the coordination model, and the agent's requested tuple space action is checked with the access control function.

6 Conclusion

In this paper we first provided a generalized coordination model representative of those required by mobile applications. We then introduced access control functions for mobile coordination and showed how it could be successfully used in systems and applications. While this new construct does incur some overhead, the expense is not prohibitive when compared with the coordination 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 National Science Foundation under Grant No. CCR-9970939 and by the Office of Naval Research MURI Research Contract No. 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 National Science Foundation or the Office of Naval Research.

References

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

[2] Bryce, C., 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* (1999), pp. 4–20.

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

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

[5] Cremonini, M., 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*, LNCS **1906** (2000), pp. 99–114.

[6] Ferraiolo, D. F. and D. R. Kuhn, *Role based access control*, in: *Proc. of the 15th National Computer Security Conf.*, 1992.

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

[8] Gray, R., 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*, LNCS **1419**, Springer-Verlag, 1998 pp. 154–187.

[9] Handorean, R. and G.-C. Roman, *Secure sharing of tuple spaces in ad hoc settings*, Technical Report WUCSE-03-26, Washington University (2003).

[10] Julien, C. 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*, 2002.

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

[12] Murphy, A. L., 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*, 2001, pp. 524–533.

[13] National Center for Supercomputing Applications, Integrated Decision Technologies Group, *SAMCat: A securable active metadata catalogue* (2002).
URL `http://idtweb.ncsa.uiuc.edu/documents/samcat.pdf`

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