Dynamic Access Control for Ubiquitous Environments

Jehan Wickramasuriya and Nalini Venkatasubramanian

Department of Information & Computer Science
University of California, Irvine
Irvine, CA 92697-3425, USA
{jwickram,nalini}@ics.uci.edu

Abstract. Current ubiquitous computing environments provide many kinds of information. This information may be accessed by different users under varying conditions depending on various contexts (e.g. location). These ubiquitous computing environments also impose new requirements on security. The ability for users to access their information in a secure and transparent manner, while adapting to changing contexts of the spaces they operate in is highly desirable in these environments. This paper presents a domain-based approach to access control in distributed environments with mobile, distributed objects and nodes. We utilize a slightly different notion of an object's "view", by linking its context to the state information available to it for access control purposes. In this work, we tackle the problem of hiding sensitive information in insecure environments by providing objects in the system a view of their state information, and subsequently manage this view. Combining access control requirements and multilevel security with mobile and contextual requirements of active objects allow us to re-evaluate security considerations for mobile objects. We present a middleware-based architecture for providing access control in such an environment and view-sensitive mechanisms for protection of resources while both objects and hosts are mobile. We also examine some issues with delegation of rights in these environments. Performance issues are discussed in supporting these solutions, as well as an initial prototype implementation and accompanying results.

1 Introduction

Security is a critical issue in mobile environments where computational entities (e.g. agents), devices and resources can be easy targets of attacks. With the rapid growth in wireless networks and mobile agent applications, security mechanisms are needed to prevent service and content providers as well as unauthorized personnel from gaining access to sensitive data and resources on mobile client devices. This paper deals with one particular aspect of security, that of access control in the presence of mobile hosts and objects. This work considers two types of mobility; *object mobility* and *node mobility*. Object mobility concerns the movement of individual objects between nodes in the network. These objects may be dynamically created within a node and migrate from node to node within

R. Meersman, Z. Tari (Eds.): CoopIS/DOA/ODBASE 2004, LNCS 3291, pp. 1626–1643, 2004.

the network, possibly carrying sensitive state information. Node mobility refers to the physical movement of mobile hosts (e.g. laptop, PDA etc.) in a distributed environment. The concept of object and node mobility is similar to the notion of logical/physical mobility [1] and virtual/physical mobility [2].

Seamless execution of secure applications in the presence of object and node mobility introduces challenges in managing mobility, maintaining concurrency and providing secure access to resources. Firstly, mobile devices have resource limitations in terms of battery power, memory and storage. Secondly, mobile hosts are subject to disconnections and varying network availability; access control mechanisms must be capable of dynamically "re-evaluating" access rights when a disconnected node rejoins at a different point in the wireless network. Thirdly, applications will need to execute seamlessly in the presence of changing access rights. Traditionally, when an application operating on secure content moves between security domains, it may need to be restarted after the user has determined what content to hide (or reacquire). One possible solution is to encrypt information as objects/nodes traverse through insecure environments. On mobile devices where resources (e.g. residual power,memory) are at a premium, additional encryption-based techniques to facilitate data hiding may be prohibitively expensive. In future wireless networks, available bandwidth between the mobile host and base station increases significantly, therefore it is possible to exploit this bandwidth to push more intelligence and computation off the device for security purposes. In this paper we explore a middleware-based approach where mobility management services at the object and node levels capture and store sensitive information before it becomes accessible in the insecure environment. Since not all state information may be accessible to an object at any point in time, a distributed, trusted repository is utilized to store sensitive information that needs to be hidden from an object. This approach is analogous to firewall-based approaches to corporate security since we "filter" out sensitive information at the object level.

To model varying security levels we adopt a multilevel security (MLS) approach [3] to facilitate sharing of data in a safe manner without the danger of "leaking" sensitive data to unauthorized users. In the MLS approach, entities are associated with different security classifications which are then used to regulate access to various resources (i.e. objects) resident on both fixed and mobile nodes. These classifications can be used as the basis to regulate access to various resources (i.e. objects) resident on both fixed and mobile nodes. In addition, we introduce the notion of the *view* of an object as the local representation of its state in its current security context. An object's view changes as it moves in and out of environments with varying security levels. We develop techniques for information hiding which utilize multilevel security, specifically taking into account the mobile nature of both *objects* and *hosts*. The remainder of this paper is organized as follows; Section 2 describes our object representation and a meta-level architecture for access control. In Section 3 we introduce the concept of domain-based access control and develop techniques for view management of objects and nodes that move through varying security domains. Section 4 exam-

ines issues relating to delegation of access rights in our framework. In Section 5, we analyze the performance of the proposed view management techniques under various mobility conditions. An initial prototype implementation of our underlying access control framework is also presented, as well as initial accompanying results. We conclude with related work and future research directions in Section 6.

2 Architecture and Object Representation

We utilize the actor model of computation [4] to represent objects. Actors provide a semantic foundation on which we build invariants to ensure safe and correct access control. In the actor paradigm, the universe contains computational agents called actors, distributed over a network and communicate via asynchronous message passing. Actors encapsulate state and a set of methods for manipulating that state, as well as a thread of control. An actor may also contain a list of *acquaintances* (other objects an actor knows about and can therefore send messages to). All entities that need to be protected are represented as actors. This includes passive entities such as files, and subsequent operations on these entities (such as read and write in the case of a file) are implemented as methods of the actor and can be accessed via message passing. As opposed to traditional access control systems where there exists a user to object relationship (e.g. users having access types such as *read* and *write* to objects) in an actor-based system, access control is enforced from the point of view of incoming and outgoing messages to a particular actor. Thus, primitive types used in traditional access control models (such as read and write) can be controlled by restricting messages that may access those methods of the actor. The system contains two kinds of actors, base-level and meta-level actors. Base-level actors carry out application-level computation while meta-level actors are a part of the runtime environment and implement system-level resource management activities (e.g. access control, migration, etc.). Meta-level actors communicate with each other via meta-level messages, however they may examine and modify information corresponding to base-level actors residing on the same node. For the purposes of this paper the specific details of the actor model are omitted but can be found in [4,5]. Throughout this paper we refer to objects and actors interchangeably.

2.1 Modelling Access Control

Multilevel security concepts are used to avoid the unauthorized disclosure of sensitive information (state, data) associated with objects. We do this by assigning *security levels* to the objects in the system, which effectively represents the level of sensitivity associated with an object (or rather the information it is carrying). This prevents access to the information if a sufficient level of clearance is not present. In addition, we model the network space as a set of federated domains or partitions, each with varying security levels (levels of sensitivity) effectively constituting access boundaries within the network. We use four levels of security to label objects and domains, but the model does not prohibit the number of

classifications that may be used. We designate the classification levels as follows; (a) Level 3 (L_3) - high-level, (b) Level 2 (L_2) - medium-level, (c) Level 1 (L_1) - low-level and (d) Level 0 (L_0) - no security. For domains, we define $dom(\alpha)$ as a function which returns the current domain of an actor α. It should be noted that if a node exists within a domain, *dom* then any actors, α in *dom* should be resident on that node (at least). Security level (SL) is a function that is applied to both objects and domains. When applied to an object α it returns the security level of α. When applied to a domain *dom*, it returns the security level of that domain:

$$SL(\alpha) \Rightarrow x, \text{ where } x \text{ is the } SL \text{ of the actor } \alpha \text{ and } x \in \{L_0, L_1, L_2, L_3..., L_N\}$$

SL_{max} refers to the maximum security level of an actor, and is usually the security level assigned at the time of creation. Furthermore, we introduce two auxiliary functions that reduce and restore the state of an object respectively. For a security level l, the function $reduce\_SL(\alpha, l)$ when applied to an object α returns the updated view for it by stripping out state and acquaintances that violate view management invariants (presented in Sec. 3). The inverse function, $reduce\_SL(\alpha, l)$ restores state information and acquaintance information corresponding to security level l. The idea is that reduction and restoration should be transparent, if the actor hasn't otherwise changed state:

$$SL(\alpha) = l' \geq l,\ restore\_SL(reduce\_SL(\alpha, l), l') \text{ then, } SL(\alpha) = l'$$

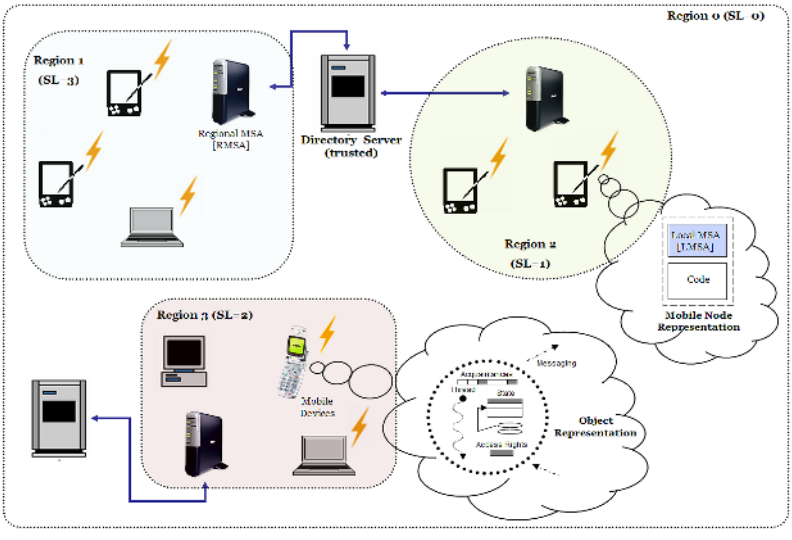

Fig. 1. System architecture: The RMSAs service the mobile devices in their region and communicate securely with the nearest directory server. The node and object representations are also illustrated. The local component of a MSA runs on each mobile device and communicates with the RMSAs in providing view management for resident objects.

2.2 System Architecture

The system environment (depicted in Fig. 1) consists of a distributed, trusted directory service (DS), which stores access control information as well as sensitive state and data associated with objects and nodes. In addition, profiles are associated with objects and nodes which include properties such as location, speed of movement and other attributes of interest which are utilized by the view management protocol (described in Sec. 3). A middleware service library is associated with the DS, whereby various core service modules can be loaded and unloaded dynamically. Mediating access to the DS is a set of Meta-security actors or agents (MSA). Functionally, there are two types of MSAs; regional MSAs (RMSA) and local MSAs (LMSA). These regional MSAs execute on distributed servers or proxies throughout the network (i.e. on a per domain basis) or on stand alone mobile hosts. RMASs are also responsible for selecting an "optimal" directory server, which may be based on various factors (e.g. location). Local MSAs execute on both fixed and mobile hosts, however in the later case the functionality is limited to a minimal set of operations in order to conserve resources. Local MSAs on fixed hosts are primarily responsible for capability verification on incoming and outgoing messages and a relevant subset of view management functionality (i.e. disconnection). The mobile environment is the focus of this paper, but further details of the local MSA implementation can be found in [6,7]. The functionality of the various meta-security entities is outlined in Fig. 2.

LMSA (non-mobile nodes)

1. Encapsulates and inteprets security policy as part of its behavior (view management, delegation/revocation etc.).
2. Mutual authentication of meta-level communication.
3. Authenticate and establish secure connections with directory services (e.g. SSL-based).
4. Capability insertion/verification for outgoing/incoming messages.
   (a) Verification that incoming messages to a particular actor have the necessary rights.
   (b) Tagging outgoing messages with capabilities (for the sender) and maintaining system-wide capability information for all objects in the DS, as well as caching this information locally.

RMSA (regional)

1. Mutual authentication of meta-level communication.
2. Authenticate and establish secure connections with directory services (e.g. SSL-based)
3. Maintain and update access control information pertaining to nodes in its domain (region).
4. Processing rights associated with messages from/to all mobile hosts in its domain, as well as routing these messages to the respective hosts (piggy-back access control information with messages).

LMSA (mobile nodes)

1. Maintain local view of objects on the node.

Fig. 2. Meta-security actor (MSA) functionality

3 View Management

In this section we illustrate how information hiding can be achieved with mobile objects and hosts. To facilitate this, we introduce the concept of a "view" of an object. Specifically, an object's view is the representation of its local state in the current security domain. An object is initialized with a view (and accompanying SL) when it is first created. Any changes to the view of an object must preserve certain multilevel security constraints. Object and node mobility can cause view changes since an object/node might move through domains with varying security levels. For example, sensitive information must be extracted from objects as they enter less secure domains. When a node changes security domains, all objects on that node must be examined for possible view changes. Several events in the lifetime of an object can affect its current view - i.e. creation, migration, disconnection of a node on which the object resides, message arrival, etc. For instance, although an object may have its local view upon disconnection, its access rights need to be re-evaluated depending on where the node "appears" next.

We define a *total view* as a representation of all objects on a given node at a given point in time. This view can be described as a subset of all state available to all objects, on all reachable hosts in the network. In other words, the total view of an object is the view at the highest security level and encompasses all the data that the object can access in the network. An object's *local view* is the state accessible to it at a given point in time. View management facilitates control over the state and data present on the mobile hosts which results in a higher degree of security for a user in a transparent fashion. In an environment where a number of users may be using the same mobile host and both the user and hardware are changing location, utilizing new services and data, we are able to overcome vulnerabilities to maintain a high level of confidentiality. Ideally, we want the host to interact spontaneously with the environment, without any direct user interaction. To achieve this we present protocols for both establishing local views of objects on mobile hosts and maintaining these views under changing mobility conditions. We utilize this view information to manipulate potentially sensitive information tied to objects and make it accessible when necessary.

3.1 A Protocol for View Management

Here we present a view management protocol that takes into account the various cases in which the view for an object can change. The protocol handles the following events; (a) *Startup*, (b) *Object Migration*, (c) *Node Migration*, (d) *Disconnection*, (e) *Reconnection* and (f) *Shutdown*. The handling of these events are outlined in Proc. 1-6. It should be noted that this protocol merely manages state information, it is the responsibility of the meta-actors to maintain individual access rights for objects on their nodes (or domains).

The function *ViewRefresh* (shown in Proc. 1) is utilized by the view management protocol in maintaining the view of a node, n. It enforces the multilevel security constraints of the domain on each object present on the host by removing or restoring state information as necessary. Note that it bases the upper

Procedure 1 : $ViewRefresh(n, dom)$

1: **for** all a in n **do**
2: **if** $SL(a) > SL(dom(n))$ **then**
3: $reduce\_SL(a, SL(dom(n)))$;
4: **else if** $SL_{max}(a) < SL(dom(n))$ **then**
5: $restore\_SL(a, SL_{max}(a))$;
6: **end if**
7: **end for**

Procedure 2 : $Startup()$

Require: Initialization of MSA
1: authenticate a secure connection to DS;
2: **for** all registered nodes, n **do**
3: $ViewRefresh(n, dom)$;
4: **end for**

bound for restoration on the SL for the object at the time of creation rather than the SL of the creator (SL_{max}). $reduce\_SL$ strips the necessary state information from the object and stores it in the directory, whereas $restore\_SL$ performs the inverse operation.

Migration: If an object migrates (Proc. 3) to a host in a domain of lower SL (in relation to the current domain of residence), the SL of that object is lowered to match the target domain. Any state information of higher SL is captured and held at the directory service. The object's local view is updated to represent this change in state. For node migration (Proc. 4) if we assume the SL of a node is n, then the objects on that node must have a SL of less than or equal to n. When a node migrates to a new domain with a lesser SL, the objects presently resident on the node are treated as a group in a manner similar to if it was merely an object moving into the new domain.

Disconnection & Reconnection: For disconnection (Proc. 5) we identify the following three types of disconnection scenarios; (a) Sign-off (shutdown); (b) Log-off (pseudo-state, where the host is disconnected, but remains registered with the directory service); (c) Failure (unexpected disconnection). Reconnection (Proc. 6) or recovery of a disconnected node (which may have failed unexpectedly) requires that it first authenticate itself with the regional meta-security agent (MSA), which then refreshes the view on the node in case of a domain change. On shutdown, the MSA on each node stores current view information (which is in memory) for all objects at the directory service.

To ensure safe interaction between domain-based access control and object and node migration the view management process must satisfy the following key invariants:

Invariant 1 (Domain Security Preservation). For all domains, every object, α must have a security level (SL) less than that of the resident domain, dom.

$$\forall \alpha \in dom,\ SL(\alpha) \leq SL(dom), for\ all\ domains\ dom \tag{1}$$

Procedure 3 : $ObjectMigration()$

1: **for** $migration\_request(\alpha, v)$ **do**
2: **if** $SL(\alpha) > SL(dom(v))$ **then**
3: $reduce\_SL(\alpha, SL(dom(v)))$;
4: **else if** $(SL(\alpha) < SL(dom(v)))\ AND\ (SL_{max}(\alpha) < SL(dom(n)))$ **then**
5: $restore\_SL(\alpha, SL(dom(v)))$;
6: **end if**
7: **end for**

Procedure 4 : $NodeMigration()$

1: **for** boundary crossing by n into dom **do**
2: RMSA authenticates n;
3: $ViewRefresh(n, dom)$;
4: **end for**

Invariant 2 (Object Creation). For all domains, we cannot create an object, α with a security level higher than that dictated by the resident domain. Furthermore the security level of an object cannot exceed that of the level assigned at creation (SL_{max}). It should noted that $SL_{cr(\alpha)}$ denotes the security level of α at the time of creation and $cr(\alpha)$ represents the object's creator. Then we require:

$$SL(\alpha) \leq SL_{cr(\alpha)}\ and\ SL_{cr(\alpha)} \leq SL_{cr(cr(\alpha))}$$

This means that if dom is where α is created, then

$$SL_{cr(\alpha)} \leq SL(dom) \tag{2}$$

which is also an upperbound on $SL(cr(\alpha))$.

Invariant 3 (Acquaintance & Message Relations). To constrain the security level of an object α's acquaintances we can utilize the security level at the time of creation, $SL_{cr(\alpha)}$;

$$SL(\alpha) \geq SL_{cr(\alpha')}\ for\ \alpha' \in acq(\alpha) \tag{3}$$

A message from an object α to object α' is deliverable only if the maximum SL of the objects in $\alpha's$ acquaintance list (α'') are less than that of the domain of α'.

$$\forall \alpha'' \in acq(msg),\ SL_{cr(\alpha'')} \leq SL(dom(\alpha'))$$

When an object α, migrates to a domain of lower security level (in relation to the current domain of residence) the SL of α is decreased to match the target domain. In addition, any of α's acquaintances of higher SL are extracted and stored at the directory service[1] before the migration process continues.

[1] The key here is that α not be allowed to leak acquaintances of higher SL to less secure domains, these acquaintances may be discarded (no restore) or held securely.

Procedure 5 : *Disconnection*()

1: **if** $node\_state = LOG\_OFF$ **then**
2: RMSA captures local view (with $TTL = x$);
3: **if** $TTL > x$ **then**
4: commit view to DS;
5: RMSA queues msgs destined for node (with $TTL = \infty$);
6: **end if**
7: **else**
8: RMSA queues msgs destined for node ($TTL = \alpha$); $\{node\_state = FAILURE\}$
9: **end if**

Procedure 6 : *Reconnection*()

1: **for** every $reconnect\_request(n)$ **do**
2: RMSA authenticates n;
3: $ViewRefresh(n, dom)$;
4: **if** ($msg\_queue(n)$! = $empty$) AND ($TTL \leq \alpha$) **then**
5: RMSA routes msgs to objects on n;
6: **end if**
7: **end for**

View Approximation. We use this technique in order to reduce the frequency of reconfiguration given rapidly changing mobility conditions. It utilizes information from the underlying location service via history-based profiles associated with both nodes and objects. The benefits of this technique stem from the fact that identification of common movement patterns allow approximation of the view to prevent excessive computation involved in constantly switching state. Both mobile objects and nodes can exhibit regular patterns of movement, or more specifically are likely to operate within a subset of the network under most conditions. An example of the latter is that of users commuting daily from home to work and back again. Consider the case when an object's migration history suggests that it is migrating among a regular set of domains. We then set the view to be the most conservative possible given the movement patterns of the object ($SL(min\{domains\})$). An issue with view approximation is that critical information may not be accessible even when a node has the required rights to access it. To address this we; (1) Trigger events on critical state information that will force a view refresh if necessary; (2) Allow the user/application to control how far the view on a host should be allowed to diverge from its base view. It should be noted that the view approximation optimizations only hold when the node/object moves in and out of the profiled domains. Any abberations from these patterns will trigger the normal restoration and reduction behavior.

4 Delegation of Access Rights

Delegation is particularly useful in mobile environments where it may be necessary to have machines with more resources perform certain computationally expensive tasks on behalf of a mobile client. Assuming authority is successfully

delegated, at some point in time the delegator may decide to revoke the granted rights. We identify two sets of problems that may arise with both node and object mobility during the delegation/revocation process; (a) It may not be possible to carry out revocation in a timely manner due to varying degrees of mobility, (b) With multilevel security domains, delegation may not be immediately possible due to the varying security constraints. Dealing with (a) is a more involved task as it requires close interaction with the location management service in order to handle revocation effectively. These solutions are outside the scope of this paper but are addressed in [8]. In the remainder of this section we address the issues arising from providing delegation in MLS-based domains (b). Existing approaches that enforce expiration as a way of invalidating delegated authority (e.g. Kerberos) may not be adequate as we may need to explicitly revoke the rights associated with an object. Alternative infrastructures that rely on certification authorities (e.g. Delegation certificates) are also unacceptable due to the object-based nature of our environment where the number of delegated (and subsequently potentially revokable) rights can be fairly large. Both techniques can also involve large computational and communication overheads which are undesirable in mobile environments where resources are at a premium.

Our delegation protocol proceeds as follows for a delegation from A (delegator) to B (delegatee). If the request originated from the same node, all that is checked is if the maximum SL of the delegated rights do not exceed $SL(cr(B))$. If so, the rights are added to B's capability list and an acknowledgement sent back to A. If the request is not local, B's MSA is looked up and the request forwarded. We allow delegation to proceed even if the security level (SL) corresponding to the current view of the delegatee actor is below the maximum SL of the rights being delegated. All that needs to be enforced is the fact that the delegatee's security level at its default view is at least as high as the maximum SL dictated by the rights. This does not cause any problems when the delegatee is at a view that falls below this maximum SL as these delegated rights will not be accessible to it. If and when it returns to a security level sufficiently high enough to utilize these rights, they will appear as part of the actor's capability list (unless the delegation has expired or been revoked). Also note that in the case where the request is not local, the delegation request may need to be propagated to various nodes until the MSA responsible for the delegatee (B) receives the request. However unlike revocation, the delay in propagating this information does not potentially compromise the security associated with the rights being delegated.

4.1 Handling Multilevel Security Constraints

We examine the compositional issues involved when delegating authority in the presence of migration and provide some insight on how they were addressed. The object migration process [9] allows objects and their associated state to move from one node to another. We denote a request by a pair (α, ν), where α is the object to be migrated, and ν is the destination node. Consider the case where an object, α is attempting to delegate a set (or subset) of its rights to another object, β.

[Case 1:] Delegator α Migrates: In this scenario, a *DelegationRequest* has been issued and the actor α wishes to migrate. It should be noted that if migration occurs first, the object in question is effectively "frozen" and these issues do not arise. Two approaches can be taken to handle this scenario; (i) Prevent α from migrating when there is an outstanding delegation request. (ii) Allow migration to proceed and then forward the acknowledgement to α's new location. Two issues arise from adopting the later approach: (a) Infinite forwarding - the actor could potentially keep migrating, growing the subsequent forwarder chain along with it and preventing delivery of the acknowledgement which terminates the delegation protocol. (b) Message delivery given the SL of the new domain - the acknowledgement may not be deliverable given α's new domain. In the case in which α migrates to a more secure domain ($SL(\alpha) \leq SL(dom)$) there is no problem, however if the new domain is less secure ($SL(\alpha) \geq SL(dom)$) the message would not be deliverable (at this point in time). To circumvent this issue our delegation protocol is altered so it routes the acknowledgement back to the MSA which maintains the acknowledgement on behalf of α. This enables the delegator to migrate to a less secure domain while allowing the delegation to proceed.

[Case 2:] Delegatee β Migrates: Here, β may migrate in the presence of a delegation. The issue here is the delivery of the *DelegationRequest* to the target β. The same issues addressed above apply here - Infinite forwarding & problematic message delivery to β in its new domain. Consider the following cases corresponding to the migration process; (i) Migration is initiated but not complete - queue the delegation request and wait for the migration to complete and then determine feasibility for delivery of the request. (ii) Migration has been completed (i.e. β -> β'). In the second case we must again consider cases in which the new domain (in which β' is resident) is of a higher or lower SL compared to the original. In the case where the domain is more secure there is no problem as before, however in the second case we encounter the same delivery problem and further migration may cause an intermediate forwarding problem if the SL reduction is non-monotonic. Note that in the case of node mobility, we can reduce the migration of all objects on the node to single (logical) group migration and use the solutions presented above.

5 Implementation and Performance

Our evaluation strategy for this paper is two-fold; we present some performance results pertaining to our prototype implementation and provide simulation results which take into account the effects of view management and maintenance. These simulation results gave us an idea of how well our schemes can perform given varying mobility conditions which are harder to evaluate in the context of an implementation. They also allow us to further tune and optimize the performance of our protocols in integrating them with our implementation.

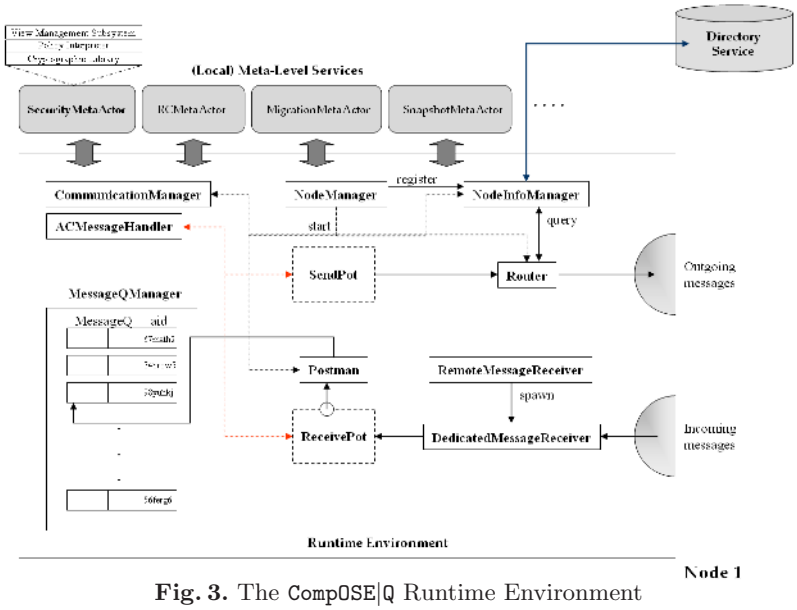

Fig. 3. The CompOSE|Q Runtime Environment

5.1 Prototype Implementation

Our prototype implementation is built on the CompOSE|Q [9] middleware architecture. This framework provides the necessary runtime semantics for the base and meta-level objects described in the meta-architectural model. The implementation of the underlying access control architecture consists of a meta-level security manager (MSA), the first-class capability objects, and a set of interfaces to the directory service (DS) which serves as a repository for object related attributes (e.g. location, access rights etc.). The system also implements a secure class loading mechanism to ensure that instantiated base-level objects are not spurious.

The CompOSE|Q framework consists of a set of runtime kernels that reside on individual nodes of the distributed system and a set of components that provide distributed systems services to the application layer. The distributed middleware layer contains meta-level services such as remote creation, distributed snapshots, DS, migration and soft real-time scheduling. The access control framework is implemented as a meta-level component within this architecture and provides protection for base-level objects. The runtime has been implemented in Java and has been tested on a variety of platforms & mobile devices. Fig. 3 illustrates how the meta-level security module is integrated with the runtime. The prototype security modules have also been implemented in Java to facilitate portability, flexibility through introspection and type-safety. This also allows capabilities to be implemented in an object-oriented manner and enables delegation of capabilities via sub-classing. We specify *capability objects* which encapsulate methods for the signing, verification, delegation and revocation of capabilities.

The Meta-Level Security Agent (MSA). Our model dictates that domain information is stored with the object (and is only readable by meta-level entities) and includes the security level of both the domain of creation (effectively SL_{max}) and the current domain of residence. The MSA encapsulates the behavior of the security service on each node, and is responsible for creation, validation & management of capabilities. At present the behavior of the MSA is defined prior to runtime but we are working on adding the ability to define customized user-policies. The main responsibilities of the MSA include: (1) Registration with the local Node Manager, (2) Installation of access rights for base-actors, (3) Message tagging, and (4) Access rights checking (enforcement). During the installation process (2), the MSA queries the Node Manager and (via the name service) obtains a list of objects present on the given node. Based on the security policy at the MSA it installs access rights for the objects (default rights if there are none specified), updates the DS with the newly defined access control information and caches the result locally. Object creation signals the MSA and the new capabilities are dynamically added to the rights table. The enforcement (4) is carried out at the middleware transport layer (See Fig. 3), which currently uses TCP sockets for messaging. The MSA intervenes during the communication pipeline via the *ACMessageHandler* component. If access control is required for a particular application, capabilities are attached to the outgoing message before it arrives at the SendPot and the message is subsequently dispatched. Likewise at the receiving end, the message is extracted from the ReceivePot, verified and delivered if access is granted.

The Directory Service. The DS is an important part of the security framework, thus optimizing performance for both storing and accessing capabilities is desirable. Our implementation of the directory utilizes the Lightweight Directory Access Protocol (LDAP) [10] and in particular the OpenLDAP group's slapd server (with the Berkeley SleepyCat DB backend). An interface utilizing the Netscape Java API was implemented that presented us with an abstraction to the set of required LDAP operations. The implemented abstraction layer allows SSL connections to the LDAP directory server. A schema representing the necessary access control information was also implemented and numerous optimizations were made to the caching policies and indexing routines to maximize performance [7].

5.2 Prototype Evaluation

The testbed for measuring the performance of our prototype utilized a network of Sun Ultra 5 workstations (333Mhz UltraSPARC IIi with 256KB external cache, Solaris 2.7, 128MB RAM) running on a 10Mb LAN. The system is implemented in Java (JDK 1.3). Performance results were obtained using JProbe Profiler 2.8. Execution times are average results over 100 iterations of the various components and are represented without JVM induced overheads.

The operational overheads for the various components of the access control subsystem are summarized in Fig. 4. *InstallRights* represents the time taken for the MSA to obtain a list of objects (local send & receive RTT) on its node and setup access rights for them (local store). On message send we measured the

	Execution Time (μs)
MSA Startup Overhead	
- Registration()	110
- InstallRights()	325
MSA Operational Overhead	
- Message Tagging* (msg send)	140
- Message Check (msg receive)	
- Verify*()	11
- Enforce()	192
Communication Overhead	
- Local Msg. Send	174
- Remote Msg. Send	239
- Msg. Receive	130
Directory Service Overhead	
- Adding Actor (w/ capabilities)	14,234
- Adding Actor Attribute	3,959
- Attribute Query	2,141

Fig. 4. Summary of performance results

raw processing overhead necessary to tag outgoing messages (i.e. from when the message is intercepted by the MSA to when it is returned to the regular messaging system). *Verify* represents the time taken to authenticate the capability. *Enforce* is the primary checking mechanism, which examines the access rights corresponding to an incoming message and authenticates it. The local message send result also reflects the overheads involved in communication between local base-actors, as well as communication between the MSA and the runtime (in particular the Node Manager). The remote message send performance depends a lot on the caching techniques (both the socket and remote actor caches) being used. A cache miss (if the Node Manager is unable to locate an actor on it's node via the cache) results in a DS query (c). The security framework overheads effectively adds to the communication overhead as the MSA simply acts as a 'filter' in maintaining access control on the node.

5.3 Simulation

The simulation consisted of modelling our framework in an environment where both objects and nodes were mobile, and where parameters relating to the mobility conditions could be varied in order to examine the effectiveness of our solutions. Mobility models [11] were implemented for both mobile nodes and objects. For the node mobility model we used a slightly modified version of a random speed-based mobility model. Furthermore we assumed the mobile nodes are initially distributed randomly in a closed coverage area which is further divided into a number of domains, each of which is associated with a security level. The nodes distributed in this coverage area are allowed to roam with a velocity v and a direction Θ with respect to the positive x-axis. We experimented with a number of well known movement patterns including random walk, a quasi-random

distribution and restricting movement of nodes to a portion of the network using a Zipfian distribution. The border rule permitted nodes to "wrap around" to the other side of the simulation plane when they hit the boundary. Mobile objects were modelled as a layer on top of this whereby they were allowed to migrate to a particular node. This was generally done in a random fashion, however each object contained a "migration list" which allowed us to define which nodes it could potentially migrate to on each move. The experiments considered the overhead of the basic view management mechanisms with respect to the DS overhead involved in maintaining both state information pertaining to objects as well as location information corresponding to the mobile nodes themselves. Domain partitioning was also varied in order to examine the impact of fine-grained vs. course grained domains on view maintenance. Security requirements were also varied to allow basic modelling of high and low security applications.

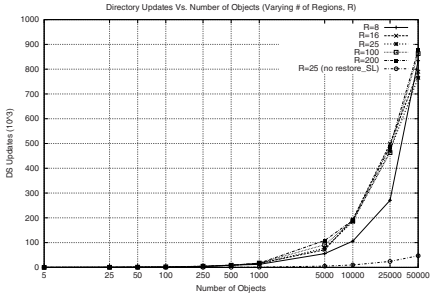

Fig. 5. Directory update overhead with varying number of objects.

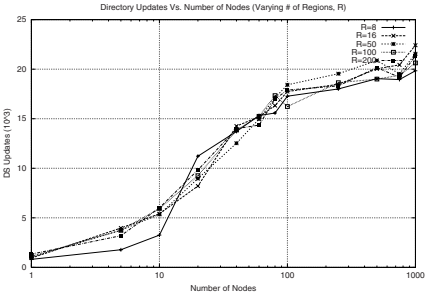

Fig. 6. Directory update overhead with varying number of nodes.

5.4 Simulation Results

Basic System Performance: Fig.5 & 6 depict the DS update overhead given an increasing number of objects and nodes. In increasing the number of objects we used a fixed number of mobile nodes, 4 security levels and a random distribution for migrating the objects. The runtime for these experiments was 300 seconds, and the number of regions was varied between 8 and 200. It should be noted that the view management algorithm implemented by each of the regional MSAs performed restoration (*restore_SL*) whenever possible for these measurements. That is, whenever an object moved into a security domain in which it's SL could be restored (up to it's maximum), depending on the domain restrictions - the restoration was performed by the MSAs. This effectively represents the maximum overhead imposed by the view management algorithm, as no optimizations were used and all updates were performed at the DS. The last plot in Fig.5 (R=25) represents a measure of the DS overhead with no restoration being performed by the MSAs. As expected the overheads are much lower and scale further in this case. In reality, a cost falling somewhere in between the two would be optimal depending on the application requirements.

As can be seen in Fig.5, the DS overhead is largely independent of the number of domains given that the number of security levels is fixed. The performance results presented here intentionally do not make any optimizations and show the characteristics of the simulation. Other results showed that simple LRU-based directory caching at the MSAs improved these times significantly. In Fig.6, the number of directory updates is significantly less when considering the case where node mobility dominates. This occurs because in the case of node mobility, the MSA performs a view refresh on the whole node in which case the directory updates are batched - making it less expensive than when individual objects are migrating constantly, crossing security domains where each update needs to processed individually.

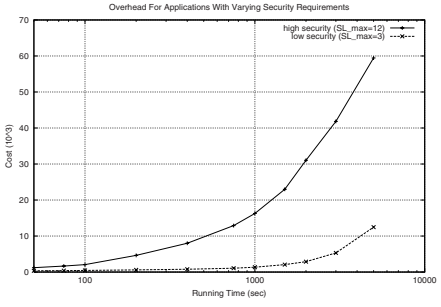

Fig. 7. Total overhead as a result of varying security requirements.

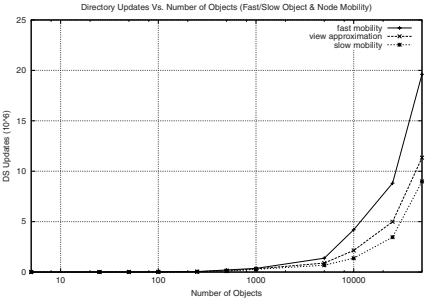

Fig. 8. Effects of view approximation with varying mobility.

Impact of View Approximation: Fig.8 examines the effect of the view approximation optimization described in Sec. 3. and depicts the DS update overhead associated with varying the number of objects in the system under varying mobility conditions. Note that as the running time increases the number of moves also increases for objects (mobile objects are migrating at approximately 3-4 times the speed of the nodes). For fast mobility, the speed at which the objects and nodes move was increased to approximately three times the normal rate and no view approximation was utilized. The second plot shows the effect of view approximation, in which we employ and maintain the most conservative view for an object when moving within its profiled domains. Here the DS overheads can be reduced by up to 40% depending on the mobility patterns dictated by the application (objects) and nodes. In contrast the third set of results shows the effects of slowing the mobility (approximately three times less than the normal case) on directory update cost. The overheads are even lower as the number of view changes are lowered due to the slow moving nature of the objects and nodes (note that the number of regions are fixed for these measurements). Fig.7 illustrates the cost involved in providing view management for a typical high security application when compared to one with minimal requirements. Cost here was modelled as the DS overheads (query and update) and meta-level messaging overheads involved in running the view management algorithm. The high security application was modelled by using unconditional SL restoration (having

the maximal state visible at all times), and SL_{max}=12 (12 levels), whereas the low security application utilized approximately 50% restoration and SL_{max}=3 (3 levels).

6 Related and Future Work

Our work spans the areas of agent-based security, view management for databases and other object-based protection schemes. The Cherubim project [12], a CORBA-compliant, agent-based dynamic security framework provides the notion of *Active Capabilities* [13] to protect and control access to the objects it is associated with. The `LIME` [14] middleware framework supports applications that exhibit both object and node mobility. OASIS [15] implements RBAC for secure, independent, internetworking services and provides a strong policy-based language for describing users and services. The main distinction of our work is the fact we consider both the mobility of hosts and underlying objects while providing access control techniques that adapt to these conditions. Meta-level access control for passive objects using a capability-based paradigm based on security-meta objects (SMO) has been explored in [16]. These meta objects are attached to object references and control access to the corresponding objects. Since objects can have many references, the overhead of such a system is potentially very high in a dynamic environment. A lot of work has been done on view maintenance for databases [17], which deals with maintaining consistent views of data in a repository in the presence of change (updates etc.). [18] deal with fine-grained protection schemes for object databases (ODBs) to provide efficient searching, browsing and processing while still maintaining control of data and good performance. In our case the view refers to a local state of an individual object, rather than a customized views of a centralized repository such as a database.

Further work is being done in developing formal semantics for access control with node/object mobility. This will help us reason about the interaction of access control and other middleware services. We are also working on using differing security policies at the meta-entities to provide for QoS requirements, power constraints etc. Additional fine-grained experiments which map various application classes' security requirements to the characteristics of the objects in the system are being utilized to categorize the types of overheads involved for some typical security requirements. The eventual goal is to build a flexible set of context-sensitive middleware-based access control techniques which can be utilized in dynamic, mobile environments.

Acknowledgements. This research was supported in part by the Office of Naval Research under ONR MURI research contract N00014-02-1-0715.

References

1. Roman, G.C., Picco, G.P., Murphy, A.: A Software Engineering Perspective On Mobility. In: Future of Software Engineering, 22nd International Conference on Software Engineering. (2000) 241–258

2. Cardelli, L.: Abstractions for Mobile Computation. In: Secure Internet Programming: Security Issues for Mobile and Distributed Objects. Number 1603 in Lecture Notes in Computer Science (LNCS), Springer-Verlag (1999) 51–94
3. Bell, D., LaPadula, L.: Secure Computer Systems: Unified Exposition and Multics Interpretation. Technical report, Mitre Corp, US (2001)
4. Agha, G.A.: ACTORS: A Model of Concurrent Computation in Distributed Systems. MIT Press (1986)
5. Venkatasubramanian, N., Talcott, C.: A Semantic Framework for Modeling and Reasoning about Reflective Middleware. IEEE Distributed Systems Online, Vol 2., No 6 (2001)
6. Wickramasuriya, J., Venkatasubramanian, N.: A Middleware Approach To Access Control For Mobile, Concurrent Objects. In: International Symposium on Distributed Objects and Applications (DOA 2002), Irvine, CA. (2002)
7. Wickramasuriya, J., Venkatasubramanian, N.: A Directory Enabled Middleware Framework for Distributed Systems. In: IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 2003), Guadalajara, Mexico. (2003)
8. Wickramasuriya, J., Venkatasubramanian, N.: Supporting Timely Revocation in Highly Mobile Environments. Technical report, Dept. of Infomation & Comp. Science, University of California, Irvine (2003)
9. Venkatasubramanian, N., Deshpande, M., Mohapatra, S., Gutierrez-Nolasco, S., Wickramasuriya, J.: Design and Implementation of a Composable Reflective Middleware Framework. In: Proc. of the 21st International Conference on Distributed Computing Systems (ICDCS). (2001)
10. Howes, T., Wahl, S.K.M.: Lightweight Directory Access Protocol (v3). IETF RFC 2251 (1997)
11. Bettstetter, C.: Smooth is Better than Sharp: A Random Mobility Model for Simulation of Wireless Networks. In: ACM Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM), Rome, Italy (2001)
12. Qian, T.: Cherubim Agent Based Dynamic Security Architecture. Technical report, Department of Computer Science, University of Illinois at Urbana-Champaign (UIUC) (1998)
13. Campbell, R.H., Qian, T., Liao, W., Liu, Z.: Active Capability: A Unified Security Model for Supporting Mobile, Dynamic and Application Specific Delegation. Technical report, Department of Computer Science, University of Illinois at Urbana-Champaign (UIUC) (1996)
14. Murphy, A., Picco, G., Roman, G.C.: Lime: A Middleware for Physical and Logical Mobility. In: Proc. of the 21st International Conference on Distributed Computing Systems (ICDCS). (2001)
15. Hine, J.H., Yao, W., Bacon, J., Moody, K.: An Architecture for Distributed OASIS Services. In: Middleware. Number 1795 in (LNCS), New York, Springer-Verlag (2000) 104–120
16. Riechmann, T., Hauck, F.J.: Meta Objects for Access Control: Extending Capability-based Security. In: Proceedings of New Security Paradigms Workshop, Langdale, Cumbria, UK. (1997) 17–22
17. Wolfson, O., Sistla, P., Dao, S., Narayanan, K., Raj, R.: View Maintenance in Mobile Computing. In: SIGMOD RECORD. (1995)
18. Bertino, E., Jajodia, S., Samarati, P.: Access Control in Object-oriented Database Systems: Some Approaches and Issues. In: Advanced Database Concepts and Resources Issues. Number 759 in LNCS, Springer-Verlag (1993) 17–44