Context-Based Secure Resource Access in Pervasive Computing Environments

Anand Tripathi, Tanvir Ahmed, Devdatta Kulkarni, Richa Kumar, and Komal Kashiramka
Contact Author:Anand Tripathi, tripath@cs.umn.edu
Department of Computer Science
University of Minnesota, Minneapolis MN 55455

Abstract: This paper presents a system architecture for supporting ubiquitous computing for mobile users across different environments by transparently performing context-based discovery and binding of resources. A ubiquitous computing environment is viewed as a collaboration space among mobile users, system services, and sensors/resources embedded in the physical environment. We present here different policies for binding resources to names in this space. We also present here a role based specification model for defining activities.

1. INTRODUCTION

With the Internet supporting ubiquitous connectivity across the globe and a growing dependence of our day-to-day activities on globally networked computing resources, a person requires his or her computing environment to be accessible from different locations. A user's activities generally include collaborative tasks in different environments involving interactions with other users and system services.

In our approach, a pervasive/ubiquitous computing environment is viewed as a collaboration space involving mobile users, system services, and embedded computing resources and sensors in the physical space. A mobile user's computing tasks are viewed as collaborative activities in this space. For this, our approach extends the middleware framework for secure distributed collaboration [11] to include support for context-based security policies for resource access by mobile users. Moreover, we allow mobile users as well as system level services to be represented as collaborating agents. We term this as *ubiquitous collaboration*.

The focus of this paper is on policies for resource access in ubiquitous and context-aware pervasive computing environments. In this paper we define the resource access models in such environments and present a specification model for ubiquitous activities. The goal of our work is to support construction of computing environments for unobtrusive mobility of users by providing secure as well as transparent access to resources and services for the user's context-based computing needs. The environment should be able to proactively discover and transparently bind the resources required by the user. The environment should be able to negotiate the security policies specified by the user and the security polices associate with the resources. Mobile users in a new environment are relieved from the burden of performing complex actions to discover resources needed in that environment. Our goals are similar to many of the current research activities in the field of pervasive computing [2, 5, 7, 8, 4].

There are several important security issues that need to be addressed towards the above goals.

- A user's access privileges for services and resources may need to be determined based on the user's role in collaborative activities. Such privileges may need to be further constrained based on the physical context of the user.
- Authentication and authorization of access based on a user's context is required for proper enforcement of the required policies.
- As a user may not fully trust a foreign environment, access as well as visibility of certain sensitive resources in the user's native environment may need to be forbidden for privacy and safety concerns. Moreover, in discovering local service/resources in an environment, the user may need to ensure that the requested service/resource is being managed by a trusted entity and meets certain security requirements.
- Similarly, an environment may need to impose restrictions on a visiting user's access to local services and resources.

An important aspect of our approach is to build ubiquitous and context-aware applications and computing environments from their high level specifications coupled with a policy-driven middleware. A number of other projects have taken this kind of approach [3, 12]. In contrast most of the other projects have used methods of custom design, implementation, and integration of services and components to realize a specific environment.

2. ISSUES IN SECURE RESOURCE ACCESS

A mobile user's access to resources in a domain are controlled by the policies defined at several different and independent levels.

- A user specifies discretionary policies controlling context-based visibility and access of his/her resources.
- A resource/service manager specifies the access control policies for mobile users. Such policies specify if the resource can be accessed from foreign domains and the context under which the access should be permitted.
- A collaborative activity specifies the policies defining conditions and context under which a user should be allowed to access a resource. Such policies may also control dynamic binding of a resource-name to a different resource when the context changes, such as the user moves from one domain to another.

All such policies need to be integrated in a coherent fashion at runtime. Associated with each user is the notion of a

resource namespace as described below:

- Name items in this space represent abstractions for various entities such as resources, services, activities, and user roles in an activity. This namespace is hierarchical as each activity defines its own namespace, and activities may be nested.
- Associate with each name item is a *descriptor* containing functional attributes and security policies required for the entity to be bound to this name.

Similar to a user's namespace, each domain also has a namespace representing various entities such resources, activities, and services. We refer to this as the *domain resource space*. The resource discovery and binding protocols bind a name item to an entity in the domain resource space. The binding of a name to a resource may change with time as the user's context changes. A user's context could be defined in terms of a number of different kinds of attributes, such as the user's current location, organizational/security domain in which the user is currently present, activities in which the user is currently participating, user's role in an activity, or devices through which the user is interacting with the environment.

A name descriptor contains the following items:

- Functional attributes expressed in schema based on WSDL (Web Services Description Language) and RDF (Resource Description Framework).
- Security and privacy policies may require that only a subset of the entities in the user's namespace may be visible to the user in a given context. This we refer to as the the user's "view" of the namespace in that context. A user's view is determined by the access control policies at various levels as noted above. The *view access control* directives in the name descriptor determine in which context the name is visible.
- Specification of context-based binding policies is given along two orthogonal dimensions as noted below:
  - The binding of a resource name could be specified either as *permanent binding* or *context-based binding*. A permanent binding never changes after initial binding is performed, unless explicitly changed by the user. A context-based binding requirement specifies the context-change events that would cause implicit invocation of resource discovery and rebinding of the name to a different resource.
  - Another binding directive would specify if a name item in a shared namespace should reflect a *shared binding* or a *private binding*. In case of shared binding, all users would access the same resource using that name. In contrast, a private binding means that for each user the discovery and binding operations are performed independently. Thus the same name may be bound to different resources.

The name descriptor could also include the domains where the resource is needed to be accessed by the user applications. This information is then used during resource discovery to ensure that the resource provider's policies permit access from those domains. Additionally, a descriptor may also specify if a resource could be cached or replicated in other domains. The name descriptors are defined to be extensible to include new policy requirements.

3. ARCHITECTURE FOR CONTEXT-BASED RESOURCE ACCESS

Here we briefly describe the system architecture for context-based resource access to support various security policies and binding requirements. This architecture is based on the Ajanta mobile agent system [10] and our middleware for secure collaboration [11].

Figure 1 shows an example of how a user's view changes when the user moves from one organizational domain to another. This figure shows two domains and their system level namespaces; it shows a user's namespace view of some activity when the user moves from domain A to B. In domain A, the user's *view* contains several resources. This figure also shows the binding policies for the resources shown in the *view*. The policy for a printer named *OfficePrinter*, given as *Permanent*, indicates that the binding should be done only once and it should not change even if the user moves to another domain. Its view access control policy, *View ACL*, specifies that that this name should be visible in all domains. The descriptor for *LocalPrinter* specifies that it should be rebinded whenever the domain changes. When the user moves from domain A to B, *LocalPrinter* is bound to the printer named B:Printers:PSPrinter. The name F1 is bound to a local file, and the caching directive causes a copy of the file to be created in domain B. The *View ACL* for the name F2 specifies that the file is visible only in Domain A.

In this architecture, a user's context-based view for an activity's namespace is managed independent of other activities of the user. This is because the user may be participating in various activities through different contexts.

In the user's environment, a *view manager* and a *context manager* are associated with each activity as shown in Figure 2. The context manager is responsible for detecting the context change events and delivering such events to the view manager. The view manager specifies the event detection policies for the context manager. The view manager is responsible for managing and updating the view based on the changes in the underlying context. The view manager interacts with the resource discovery services to find the resources matching the requirements in name descriptors.

Figure 2 illustrates the different ways in which the context underlying a view could change, thus causing view updates and rebinding of names. The user is executing an application on a desktop, shown as *View 1*. He now moves this application environment to a laptop. The context manager informs the view manager of this "device context" change. The view manager updates the view, based on the context-based binding policies, to rebind certain names. The new view is shown as *View 2*. Now suppose that the user moves from domain A to B along with this laptop. Once again the view manager is informed of such changes by the context manager.

Our architecture utilizes mobile agents for context-based view management functions for low power devices to conserve their resources. The view management task is offloaded from the device to an agent executing on some host in the infrastructure. This is shown in Figure 2 *View 4*.

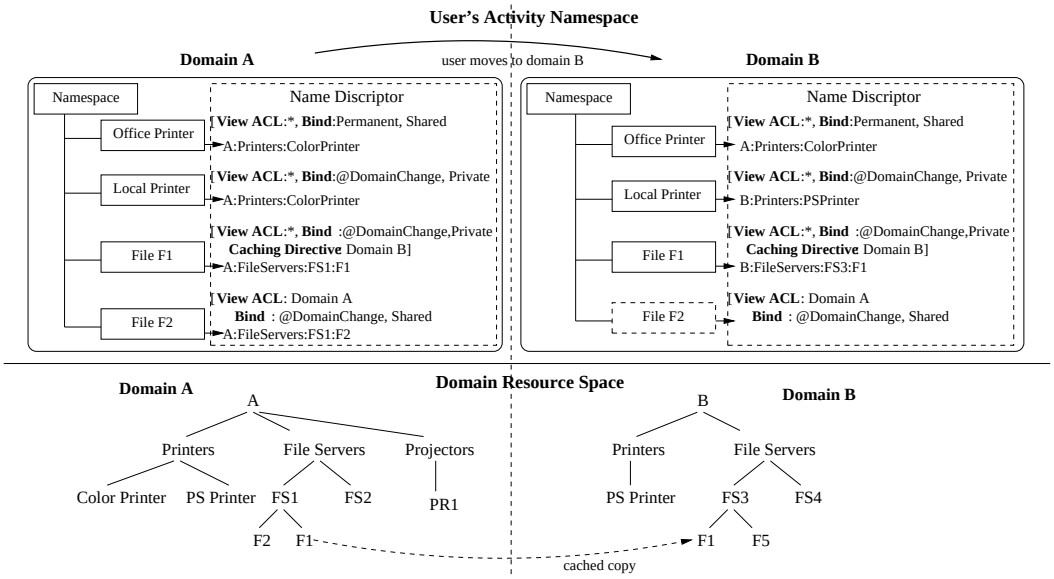

Figure 1: Namespace Mobility

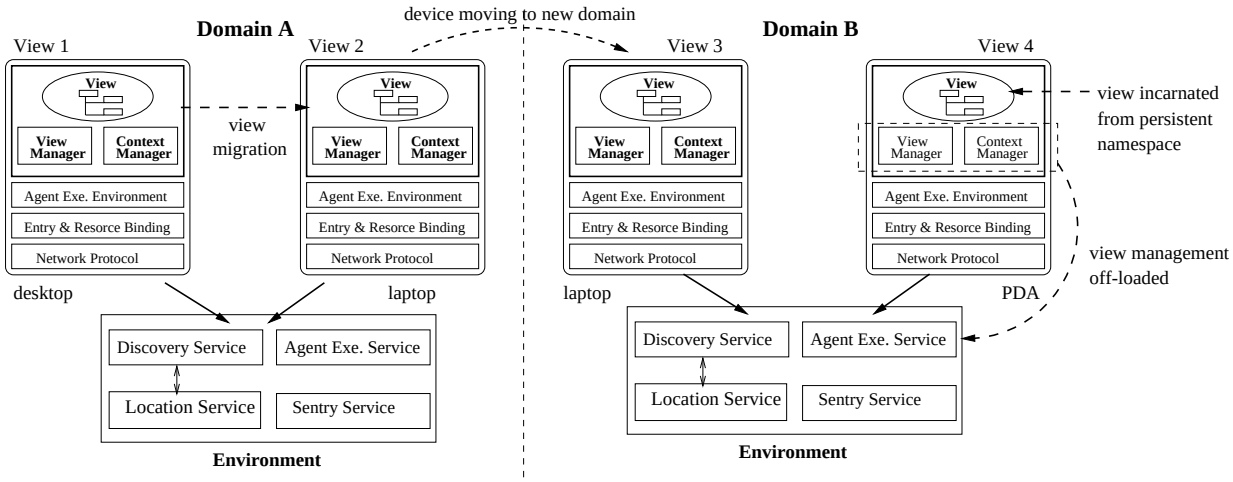

Figure 2: View Management

4. SPECIFICATION MODEL FOR UBIQUITOUS ACTIVITIES

We have developed a role based specification model for activity and security specification in ubiquitous collaboration environment. In our specification model, coordination and security policies are specified using roles. In our earlier specification model [11] for collaboration systems, we have extend traditional RBAC [6] to support specification and enforcement of various security policies such as "separation of duties" and dynamic access control. The specification model is extended for ubiquitous environment to include concepts such as active entities, resource descriptions, resource bindings, and view access control. Our roles are defined in the context of an activity. Examples of context sensitive roles include Team Based Access Control(TMAC) [9]. Like others [1], roles in our model are not limited to represent privileges, tasks, or obligations of human users but also those of active entities interacting with ubiquitous environment, such as a room-controller.

Activity Specification: In our specification model, an *activity*, which is an abstraction of a collaboration session, provides a protection domain and scope for roles, objects/resources, and privileges. An activity can be structured hierarchically, consisting of multiple nested concurrent activities. Objects can be passed into nested activities and users in roles from a parent activity can join roles in nested activities.

An *activity template* specifies a reusable interaction pattern among a set of roles using some shared objects. Activities are instantiated from templates. Besides the roles in an activity, there are several meta-roles, such as *creator* and *owner*, to manage activities.

Role Specification: Users and other active entities in an environment are represented by their roles, and roles are assigned privileges to perform certain tasks. We term these role specific tasks as *operations*. Operations can be method invocations on shared objects, synchronization actions, or

management related actions. A role operation can also result in a sequence of interactions between the invoker and a set of shared objects. This represents a *session* in the context of that role operation. Role operations may have *preconditions* to coordinate users' actions. Users acquire privileges to perform tasks in the collaboration by joining or being admitted to roles. For roles with human users, role *admission constraints* specify the conditions that need to be satisfied when a user joins a role. When a role operation's precondition is satisfied, a member in that role can invoke that operation.

To support the construction of *smart environments*, a variant of the *operation* construct called *reaction* is defined. In contrast to an operation, whose execution involves interaction with one of the participants in a role, a *reaction* is executed spontaneously when its precondition becomes true. Even for the roles which represent human entities, the execution of a reaction does not involve a human user unless specified. In such cases, a reaction executes on behalf of the role owner.

Event Specification: Events and event counters are used for specifying interactions, context changes, and security requirements. Events fall into three broad categories. The first is the class of events that are defined by the application programmer. Such events are explicitly signaled in a role operation or reaction using the *NotifyEvent* primitive. The second category of events are implicitly generated by the system, and these events correspond to instantiation of activities, execution of role operations, admission of users in roles etc. Related to each operation are three types of events: *request*, *start,* and *finish*. Our model also facilitates inclusion of application-specific data, in the form of a generic *context* object associated with each such event. The third category of events correspond to the underlying physical and runtime environment, and they are used for constructing "smart" environments. Examples of such events include user-presence detection, context changes and notification of resource utilization status. A specification explicitly declares such events being "imported" from the underlying middleware supporting the runtime environment.

In the specification model, multiple occurrences of a given event type — such as multiple executions of an operation — are represented by a list. The expression *(eventName)* returns the list including all the instances of this type of event. We provide a count operator # on lists. One can also define a filtering predicate on an event list based on certain event attributes to obtain a subset of the events. For example, for a role operation execution, we can define a filter based on invoker id, such as opName.start(invoker=Alice).

The *OfficeLockManager* as shown in Figure 3 provides an example of our event based policy specification. In the example, a smart office environment defines a *OfficeLockManager* role for automatically opening and locking the door to the office everyday at a specified time, after ensuring that no employees are still in the office. To facilitate this, environment events *EmployeeArrival* and *EmployeeDeparture* are imported from the underlying physical system.

Example Specifications: We use two examples to illustrate some of the requirements of ubiquitous collaboration environments. Our specifications are expressed in XML. However, in this paper we use a notation that is simple to read and conceptually easy to follow.

• Figure 4 shows a *Meeting* activity template, containing three roles: *Accountant*, *Manager* and *Staff*. The activity needs a *Room* object which provides access to certain resources within the meeting room such as the *projector*, *printer*, and *light*, and also supports queries to detect user presence. A *DisplayFinancialData* operation is defined for the *Accountant* role, which allows the accountant to display the financial data on the projector's display. The operation's preconditions ensure that the accountant can perform this operation only when the participant and a manager are present in the meeting room. The function *members(role)* is a role membership function, which returns the list of members in a role.

```
Import:  EmployeeArrival, EmployeeDeparture;
Role OfficeLockManager
  Reaction LockDoor
    Precondition:
      #(EmployeeDeparture) - #(EmployeeArrival) = 0
      & time > 16:30:00
      & #(OpenDoor.finish) - #(LockDoor.start) > 0
  Reaction OpenDoor
    Precondition:
      #(EmployeeArrival) - #(EmployeeDeparture) > 0
      & time > 8:30:00
      & #(OpenDoor.start) - #(LockDoor.finish) = 0
```

Figure 3: Specification for OfficeLockManager

• Figure 5 represents a *RoomController* activity to model a smart room environment. The *LightController* role is defined to automatically switch the lights on/off based on presence of users in the room or dim the lights when there is a user in the room and the *projector* is on. The appropriate *Room* object is bound to the activity at the time of its creation to provide a access to the *light* object.

The above two are independent activities, however they may share the same *Room* object.

Context-Based Security Policies: In the specification model, environment related queries are used to specify context sensitive coordination and security policies, e.g. detection of presence of a specific user in a room. Similarly, events related to the changes in a user's context, such as those related to domain, device, location, and network connection support context sensitive policies. In Figure 4, to detect that the accountant and a manager are present in the meeting room when the accountant tries to display the financial data, the room object queries for user presence information. Moreover, by computing the intersection of the *Manager* role's members with the set of all users present in the room, the presence of a manager is detected.

Resource Description: To be able to use the same activity template in different environments, the specification model includes a description of the environment related resources needed by the activity. Such descriptions are provided in XML based on WSDL and RDF. These include the following: (1) The desired attributes and components of the resource. For example, in Figure 4 the template for the *Room* object provides the necessary components of the object, based on which the resource is discovered and bound when the activity is created. (2) A *View ACL* which lists all domains in which the resource should be visible in the user's namespace. (3) *Caching Directives* - that list the domains where the resource, if visible, could be cached.

From the activity specification, we derive namespace descriptors. The view mangers associated with the users' envi-

```
ActivityTemplate Meeting {
  ObjectType Room {
     Components:
        Projector projector {...}
        Light light {...}
        Printer printer {...}
     Methods:
        Boolean isPresent(userId)
        List presentUsers()
  } as room;
  Role Accountant {
     Operation DisplayFinancialData {
        Precondition
           room.isPresent(thisUser)
           & #(room.presentUsers()
              ∩ members(Manager)) > 0
     Action projector.display(data)
  }
  Role Manager {....}
  Role Staff {....}
}
```

Figure 4: Context aware collaboration activity

```
ActivityTemplate RoomController
                 (ObjectType Room room) {
  Role LightManager {
     Reaction SwitchOnLight {
        Precondition
           #(room.presentUsers()) > 0
        Action room.light.switchOn()
     }
     Reaction SwitchOffLight {
        Precondition
           #(room.presentUsers()) = 0
        Action room.light.switchOff()
     }
     Reaction Dim {
        Precondition
           #(room.presentUsers()) > 0
           & room.projector.on()
        Action room.light.setLevel(LOW)
     }
  }
}
```

Figure 5: Smart Room activity

ronments perform the required resource discovery and binding functions.

5. CONCLUSIONS

The work presented in this paper extends our previous work on specification-based construction of secure collaboration systems to support ubiquitous and pervasive computing environments. In this paper, we have presented a model for secure resource access in pervasive computing environments based on different levels of policies. This work demonstrates that the notion of collaborative activities is a natural abstraction for specifying and building pervasive computing environments.

Acknowledgment This work was supported by National Science Foundation grant ITR 0082215.

6. REFERENCES

[1] Michael J. Covington, Wende Long, Srividhya Srinivasan, Anind K. Dev, Mustaque Ahamad, and Gregory D. Abowd. Securing context-aware applications using environment roles . In *Proceedings of the sixth ACM symposium on Access control models and technologies* , pages 10–20, May 2001.

[2] David Garlan, Dan Siewiorek, Asim Smailagic, and Peter Steenkiste. Project Aura: Toward Distraction-Free Pervasive Computing. *IEEE Pervasive computing*, 1(2):22–31, April-June 2002.

[3] Christopher K. Hess, Manuel Roman, and Roy H. Campbell. Building Applications for Ubiquitous Computing Environments. In *International Conference on Pervasive Computing (Pervasive 2002)*, August 2002.

[4] Tim Kindberg and Armando Fox. System Software for Ubiquitous Computing. *IEEE Pervasive computing*, 1(1):70–81, January/March 2002.

[5] MIT Project Oxygen. Available at url http://oxygen.lcs.mit.edu/.

[6] Ravi Sandhu, Edward Coyne, Hal Feinstein, and Charles Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, February 1996.

[7] M. Satyanarayanan. Pervasive computing: vision and challenges. *IEEE Personal Communications*, 8(4):10–17, August 2001.

[8] Bill Schilit, Norman Adams, and Roy Want. Context-Aware Computing Applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, pages 85–90, Santa Cruz, CA, US, 1994.

[9] Roshan K. Thomas. Team-based access control (TMAC): a primitive for applying role-based access controls in collaborative environments. In *ACM Workshop on Role-based Access Control*, pages 13 – 19, 1997.

[10] A. Tripathi, N. Karnik, T. Ahmed, R. Singh, A. Prakash, V. Kakani, M. Vora, and M. Pathak. Design of the Ajanta System for Mobile Agent Programming. *Journal of Systems and Software*, 62:123–140, 2002.

[11] Anand Tripathi, Tanvir Ahmed, and Richa Kumar. Specification of Secure Distributed Collaboration Systems. In *IEEE International Symposium on Autonomous Distributed Systems (ISADS)*, pages 149–156, April 2003.

[12] Stephen S. Yau, Fariaz Karim, Yu Wang, Bin Wang, and Sandeep K.S. Gupta. Reconfigurable Context-Sensitive Middleware for Pervasive Computing. *IEEE Pervasive Computing*, 1(3):33–40, July/September 2002.