



# A specification model for context-based collaborative applications<sup>☆</sup>

Anand R. Tripathi\*, Devdatta Kulkarni, Tanvir Ahmed

*Department of Computer Science, University of Minnesota, Minneapolis, MN 55455, United States*

---

## Abstract

This paper presents a model for building context-based systems in pervasive computing environments from high level specifications. A pervasive computing environment is viewed as a collaboration space consisting of mobile users, system services, and sensors and resources embedded in the physical environment. The approach presented here is developed for building collaborative activities in which users and environment services cooperate towards some shared objectives and tasks. The specification model allows expression of policies related to context-based discovery and secure access of resources, and role-based interactions among users and environmental services. Using several examples we illustrate the capabilities of our specification model for expressing various kinds of context-based requirements for resource access and user interactions.

© 2005 Elsevier B.V. All rights reserved.

---

## 1. Introduction

An important requirement for pervasive computing environments is to support seamless mobility of users across different physical spaces and locations. A user's computing activities typically involve collaborative tasks with others in office environments, business transactions, personal matters, or entertainment activities, while frequently moving across

---

<sup>☆</sup> This work was supported by NSF grant 0411961.

\* Corresponding author. Fax: +1 612 625 0572.

*E-mail addresses:* [tripathi@cs.umn.edu](mailto:tripathi@cs.umn.edu) (A.R. Tripathi), [dkulk@cs.umn.edu](mailto:dkulk@cs.umn.edu) (D. Kulkarni), [tahmed@cs.umn.edu](mailto:tahmed@cs.umn.edu) (T. Ahmed).

different computing domains. The nature of interactions can range from ad hoc and unstructured to highly structured and coordinated. Moreover, many times a group of individuals or autonomous organizations may need to form virtual organizations to work together on some collaborative activities. In such activities, the users frequently share resources provided by other participating individuals and organizations, and they create new ones. User privileges to access shared resources and perform tasks generally depend on their roles [1] in the activity as well as their current context [2]. The context may be based on a user's physical location, computing environment, computing devices used by the participant while performing a task, coordination requirements, and any temporal constraints.

One can also view interactions between users and services in a pervasive computing environment as a form of collaborative activity. We view a pervasive computing environment as a collaboration space involving mobile users, environment agents representing system services, and embedded computing resources and sensors in the physical space. Mobile users' computing tasks are viewed as collaborative activities in this space. We call this *ubiquitous collaboration*. For building such computing environments, we have extended our previous work on a specification-based framework for constructing secure distributed collaboration environments [3,4] by including support for context-based security policies for resource discovery and access by mobile users.

An important aspect of our approach is to build context-based collaborative applications in pervasive computing environments from their high level specifications. This approach is supported through a middleware [5] that we have developed for integrating users, application-defined components, and infrastructure services to build the runtime environment of a collaborative application. The primary focus of this paper is on the specification model which we use to specify various context-based requirements for resource access, security, and coordination in such environments. These requirements include the following: specification of resources and services required in an activity, dependencies of these requirements on context information, directives for context-based resource discovery, and specification of dynamic security policies and coordination requirements. We briefly present in this paper the middleware components that enforce various policy aspects of the context-based security and coordination requirements.

The specification model presented here provides a composition framework for building context-based collaborative activities by dynamically integrating users, applications, and environmental resources. This model is based on the notion of roles for enforcing security and coordination policies. One can express context-based policies that may depend on either the internal context of the activity, such as the history of interactions among users, or the external context, such as the physical location of a participant in the activity. This model provides a variety of primitives for context-based binding of resources in an activity. Using external context information one can express policies for location-based resource access by mobile users. Moreover, this model also supports the definition of autonomous activities for implementing environmental agents and services in a "smart environment".

In Section 2 we present a model for pervasive computing environments in which user activities are viewed as collaborative interactions. Section 3 describes our specification model. In Section 4, we illustrate the capabilities of our model through several examples of collaborative activities in pervasive computing environments. The main components of

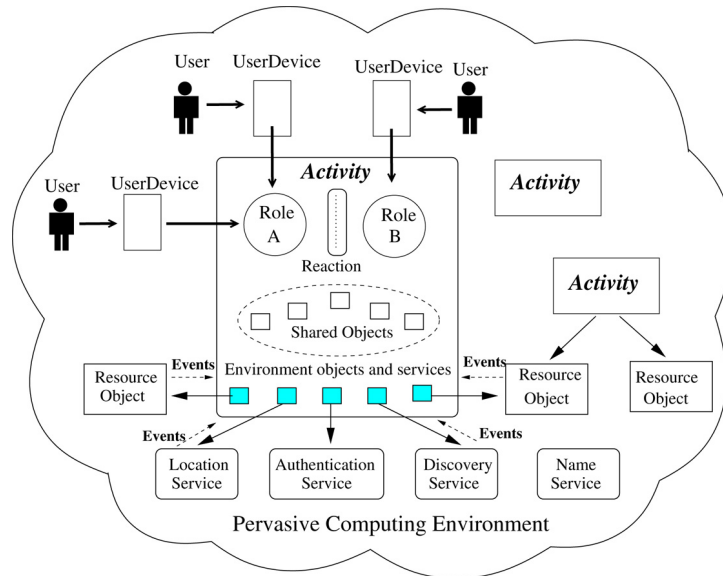


Fig. 1. An activity-based view of pervasive computing environments.

the middleware to realize a collaboration activity from its specification are presented in Section 5. In Section 6 we discuss the related work, and we present our conclusions in Section 7.

## 2. A model for pervasive computing environments

We view a pervasive computing environment as an interaction space among users and services within an environment. The computing tasks within such an environment involve interactions among users and the environmental services. We term these computing tasks that are performed symbiotically by the users and the environmental services towards a common objective as *pervasive activities*. In our framework the notion of an activity represents a broad range of interactions between users and environmental services. Many such pervasive activities may exist in an environment. Some activities may be defined to execute permanently in the environment to perform certain environment related functions, without requiring interactive participation of any user.

The examples of a pervasive activity can include a visitor touring a museum and getting information about displayed items in the vicinity, a business meeting involving participants located at different physical locations and some of the participants occasionally moving from one location to another, a medical procedure involving a patient, nurses, doctors, and other technicians in the hospital, or an activity controlling building security.

Fig. 1 shows the various elements in our model of pervasive computing environments. Users can create new activities within an environment, or they can join some existing activities based on their security privileges. Users participate in an activity by joining a

role and executing role related operations. A role represents authorization of its members to invoke a set of operations representing tasks in the collaboration space. Therefore, admission of users into a role needs to be controlled according to the security policies, which may be based on context-based constraints. Furthermore, a role operation may have a precondition to enforce policies for coordination and dynamic security. Such preconditions may also be context-based.

Users in a pervasive computing activity share and interact with resources and services that are either acquired from the environment or created within the activity itself. An activity defines a namespace for such shared resources and services, and specifies their functional descriptions. An environment typically consists of some commonly needed services such as name service, authentication service, discovery service, location service, and other services to manage activities. Additionally, an environment may also include other services that are abstractions for resources in the physical space. For example, the service may provide an abstract representation of a meeting room with certain devices such as projector, printer, and sensors situated in that room.

Within an activity, the policies for coordination among participants as well as binding and access control of shared resources/services may depend on context-based [2] requirements. We classify context into two categories: *internal* and *external*. The internal context is related to the execution state of various tasks in the activity. External context represents the attributes that are related to the physical environment. A user's external context may be defined in terms of a number of different kinds of attributes, such as the user's current location, the organizational or security domain in which the user is currently present, applications in which the user is currently participating, or devices through which the user is interacting with the environment. The location attributes may be defined in many different ways, such as GPS, presence in a building or room, proximity to certain devices or users, or the Internet domain. The external context information may be obtained by an activity from various services provided by the environment. An activity may either query a service or subscribe from it events representing external context information.

Below we present a number of examples of pervasive computing activities to highlight important requirements related to context-based resource access and security policies. In later sections we demonstrate how the primitives provided by our specification model support expression of these requirements.

#### *Meeting room scenario*

Consider a business meeting involving participants in different roles such as *Chairperson*, *Manager*, *Accountant*, and *Secretary*. A meeting may exhibit several context-based requirements for resource access, coordination, and security. A meeting room with a projector and a printer has to be made available to the participants of the activity. This room and its contained equipments would be modeled as environment resources and suitably bound to the various entities in this activity's object namespace. Such binding operations would be realized transparently. Thus, a meeting task for displaying a presentation data would use the projector in the meeting room. Similarly, a print operation executed by any participant would be directed to the printer in the room. These reflect some of the context-based resource binding requirements of the activity.

During the meeting, the participants in the meeting may need to access some existing business documents archived in the environment. A location-based security policy may require that a participant is allowed to access sensitive documents only when both the participant and his/her computing device are situated in the meeting room. One may even further require that such location-based security policy be enforced on all aspects of the meeting, and that too for all participants. Similarly, constraints may also need to be specified based on contexts of different participants in the meeting. For example, the operation of displaying financial data on the projector may be allowed only when at least one member from each of the *Chairperson* and the *Accountant* roles are present in the meeting room.

The meeting activity may allow the participation of only the users with some specified credentials (e.g., the users who are valid employees of the business unit). The admission of users into different roles in the activity needs to be properly controlled according to such requirements. It may also require that role admission constraints are context-based. For example, a person can be admitted to the *Secretary* role only when present in the meeting room and if a person in the *Manager* role is also present in the same room.

#### *Distributed meeting*

The above business meeting scenario may be extended to where the participants are present in different rooms, possibly at different sites. Such a distributed meeting brings forth additional requirements. For example, a member in the *Accountant* role can perform any of the role operations only when the person in the *Chairperson* role is also present in the same room. A further extension of this policy might require that the presentation performed by an accountant should only be displayed in the room where the accountant is present. On the other hand, presentations performed by the person in the *Chairperson* role should be displayed in all of the meeting rooms. These policies illustrate resource binding requirements which depend on the location-based context of the participants in various roles.

#### *Context-based museum information system*

Some activities may need specific data or information about a participant's personal preferences to be used in resource access primitives. For example, a visitor touring a museum obtains the audio commentary about the items displayed in the room where the visitor is currently present. The visitor may set the preference for a language for the audio commentaries. The visitor's listening device would need to be bound to the audio stream for the preferred language. Other preferences may further indicate choices for different kinds of commentaries, such as historical aspects, artist's personal information, or the artistic aspects of the displayed items.

#### *Patient monitoring*

Patient health-care activities exhibit a rich set of context-based security and resource access requirements. Several roles are involved here: patient, physician, specialist, surgeon, nurse, pharmacist, and insurance agent. Consider the following requirements in a patient

---

```

Activity activityName
  { Parameter objName }
  { Object [ Collection ] objName RDD rddSpec }
  { Bind Binding-Definition }
  { Reaction Reaction-Definition }
  { Role Role-Definition }

```

---

Fig. 2. Activity syntax.

monitoring activity. A nurse is allowed to access a patient's record only when close to the patient's bed. Moreover, a patient's records should be accessible through the nurse's PDA only when the nurse is present close to the PDA. For these requirements, we need to express resource access policies that are dependent on the device-based context of a user.

### 3. Specification model for context-based collaborative activities

In our collaboration model, an *activity* defines how a group of users cooperates toward some common objectives by performing tasks involving shared resources and infrastructure services. In an activity, users are represented by their roles, and roles are assigned privileges to perform certain tasks. Shared resources and services are represented as objects in the collaboration space defined by an activity. An activity defines a scope for user roles and shared objects, and specifies policies for their interactions.

Based on the resource requirements of an activity, certain resources/services may be discovered in the environment and bound to the objects defined in the activity. Moreover, objects may also be created within an activity or passed as parameters to it. An activity may define its resource-binding requirements to be context-based.

A *role* defines a set of *operations*. A role operation may involve execution of some actions on objects defined within the activity. A role operation can only be invoked by a member in the role. A role operation can have a precondition that must be satisfied when the operation is executed. Context-based access control policies and coordination constraints are specified as operation preconditions. Within an activity some operations may need to be automatically executed when certain conditions become true. Such operations are termed *reaction*.

We have developed an XML schema for pervasive activity specifications. Here, rather than using XML, we use a notation that is conceptually easy to follow. In Fig. 2, the syntax for the XML schema for *activity* definition is shown, where [ ] represents optional terms, { } represents zero or more terms, | represents choice, and boldface **terms** represent tags for elements and attributes in XML schema.

Fig. 3 presents the outline of a partial specification of a *Meeting* activity. When instantiated, a *Meeting* activity would need to be bound to a room described by the *Required-Room-Description* RDD. The *Meeting* activity has three roles: *Chairperson*, *Accountant*, and *Secretary*. In the following subsections different elements of the activity specification are explained in more detail through examples.

```

Activity Meeting {
  Object room RDD Required-Room-Description
  Bind room With discover (floor=5, capacity > 5)
  Role Chairperson {...}
  Role Accountant {...}
  Role Secretary {...}
}

```

Fig. 3. Skeleton specification of a *meeting* activity.

---

```

RDD [name]
  { RDD ... }
  Interface { Interface Definition }
  Attribute { AttributeName AttributeValue }
  [EventSubscription { EventName } ]

```

---

Fig. 4. RDD syntax.

### 3.1. Object namespace within an activity

The object declaration in an activity defines a namespace for the objects which represent resources and services that are required in the activity. These are either acquired from the environment or created during the life of the activity. There are two kinds of namespaces defined within an activity. The first kind defines names for objects that are shared by all participants in the activity. This is called *activity namespace*. These activity-wide names are visible in all the role operations inside an activity. The other kind of namespace defines names for objects that are private to each member in a given role. This is called *role-member namespace*. Such a namespace is managed separately for each member of the role. When a new member joins a role, a new namespace for that member is created.

A name within the namespace can refer to either a single object or a collection of objects. Objects from a collection can be selected based on various attributes of the member objects.

### 3.2. Specification of objects and services

For describing resources and services, we have developed an XML schema, termed Resource Description Definition (RDD). RDD is similar to a combination of RDF (Resource Description Framework) and WSDL (Web Service Definition Language). The main elements in an RDD are the attribute-value pairs, the interfaces, and the events that are exported by the resource. The structure of an RDD specification is shown in Fig. 4. The RDD of a resource can contain RDDs of other resources that are contained in the resource being specified.

There are two functions for which RDDs are used in our specification model. First, the capabilities of resources and services are described by their providers using an RDD. The resource providers register such descriptions with the discovery services. Second, a

```

RDD Room-Provider {
  RDD Projector {
    Interface
      void displayData(data)
    Attribute }
  RDD Light {
    Interface
      switchOn()
      switchOff()
      setLevel()
    Attribute }
  Interface
    Boolean isPresent(userId)
    Boolean isAnyPresent(userIds)
    List presentUsers()
  Attribute
    building=Computer-Science
    floor=5
    roomNumber=212
    capacity=25
  EventSubscription
    statusChangeEvent }

```

Fig. 5. RDD of a room provider.

pervasive activity describes its resource requirements through an RDD. The provider RDD is detailed and contains a complete specification of the resource/service. The RDD used in specifying the activity requirements is matched with provider RDDs. During a resource discovery request, certain fields in the requirement RDD may be filled based on some user context information. The *discovery service* returns a URL of a service or codebase of the resource.

An example of a provider RDD is shown in Fig. 5. This room generates events of type *statusChangeEvent* whenever there is any change in the room's state, such as users entering or leaving the room, data being displayed on the projector, or the light being turned on/off.

The room requirement for the *Meeting* activity is presented in Fig. 6. This *Required-Room* RDD is matched with the provider RDDs. This RDD specifies that the room in the *Meeting* activity should have a projector with an interface to display data and a light which can be turned on. Also the room resource should provide an interface method to query the presence of a particular user in the room and also a method to return the list of users present in the room. In this example, the room is assumed to be equipped with a certain set of sensors (such as RFID tags and sensors) and all users and their tags are known.

### 3.3. Specification of context-based conditions

Preconditions are specified for role operations, admission, and activation to enforce required coordination and security constraints. Such conditions are expressed in terms of predicates based on events occurring within the activity, role memberships of

```

RDD Required-Room {
  RDD Projector {
    Interface
      displayData(Data)
    Attribute }
  RDD Light {
    Interface
      switchOn()
    Attribute }
  Interface
    Boolean isPresent(userId)
    List presentUsers()
  Attribute
    building=Computer-Science
  EventSubscription
    statusChangeEvent }

```

Fig. 6. RDD of a room requirement.

participants, and query methods of the environmental resources representing external context information.

There are two categories of events, *internal* and *external*, that are utilized in our specification model. Internal events are generated by execution of operations and reactions within an activity. Specification of conditions based on internal events was developed in our earlier work [3]. Internal events are generated as part of the execution of operations and reactions. These events are represented by the names of the corresponding operation or reaction. Each operation event has two predefined attributes: *invoker* and *time*. All the events related to previously executed operations and reactions represent an event list. The specification model supports various functions on event lists. The count-operator # returns the numbers of events in a list, and a sublist can be obtained by applying a selector predicate. For example, the expression #(*opName(invoker=member(Chairperson))*) returns the number of times a member of the *Chairperson* role has invoked the operation called *opName*.

External events are generated by the objects in the environments. Such events are used in a specification to capture policies that depend on the external context. For example, an activity can specify execution of resource binding directives or reactions when certain events occur in the environment. Examples of such external events include user-presence detection, changes in the physical environmental state, and notification of resource utilization status. A specification explicitly declares, as part of object requirements, that such events are being “imported” from a service. In Fig. 6, the *statusChangeEvent* is declared to be an event imported from the required room object.

To express preconditions that depend on the context information represented by an object in the environment, a condition can include functions which query the object state. These query interfaces are also declared as part of the RDD. For example, in Fig. 6, the method *isPresent(userId)* supported by a room object can be invoked to check if a specific

user is present in the room. Preconditions can also include functions that query a user's membership in a role.

### 3.4. Role

A role is an abstraction as well as a mechanism for specifying and enforcing users' privileges in an activity. A role defines a set of *operations* that can be executed by its members. The admission of users as members in a role and invocation of other operations by the members is required to be controlled in order to enforce required security and coordination policies. These policies are specified in the form of preconditions associated with role operations and admission.

Some preconditions may depend on the context and the previous operations executed by the role member who is currently invoking an operation. We use the pseudo variable `thisUser` in the specification model to identify such a role member. Moreover, to express constraints based on the context of the device through which a user is currently participating in the activity, we use the pseudo variable `thisUserDevice` to represent the device of the role member invoking an operation. Devices can have differing capabilities. Some devices may have Bluetooth and infrared communication facilities along with IP based network connectivity. Some devices may provide proximity sensing capability. All such device specific details are abstracted in our specification model by representing the user's device by the object `thisUserDevice`.

Associated with each role, there are two types of constraints that are imposed on all role members:

- *Role admission constraints* must be satisfied when a user is to be admitted to a role. These constraints can be based on the history of the operations previously executed, the context of the user, such as the user's membership in other roles, or state of an object. For example, in the *Meeting* activity only a person present in the *Manager* role can be admitted to the *Chairperson* role. Moreover the person has to be physically present in the meeting room to be admitted in the *Chairperson* role.
- *Role activation constraints* must be satisfied for performing any role operations and reactions. Similar to role admission constraints, activation constraints can be based on event history as well as user and object context. For example, in the *Meeting* activity, the *Secretary* role can perform operations only if members of both the *Manager* and the *Accountant* roles are present in the meeting room.

A boolean function `member(thisUser, roleId)` in the precondition of a role operation checks if the user invoking the operation is present in the specified role. The function `members(roleId)` returns the role member list. Set operations can be performed on role member lists. A *count* operator, `#`, can be applied on a member list. The count of the members in a role is `#(members(roleId))`. Moreover, within a role, one can refer to it as `thisRole`. Fig. 7 presents the syntax of a role definition. The objects declared within a role represent a separate namespace created for each member in the role, and binding of these names is performed independently for each member.

The role admission constraints for the *Chairperson* role are presented below. There are three constraints in this example: (1) there can be only one member in the *Chairperson*

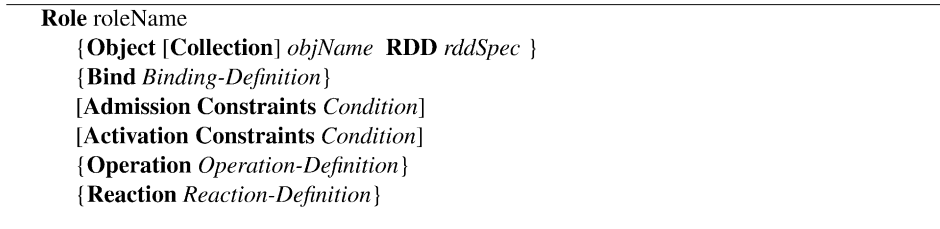


Fig. 7. Syntax for role definition.

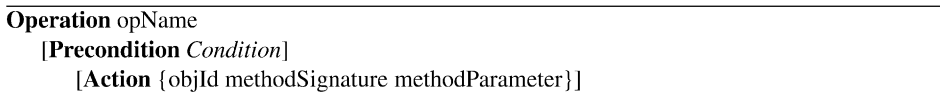


Fig. 8. Operation syntax.

role, (2) at the time of joining this role, the user must be present in the meeting room, and (3) only a member of the *Manager* role can join this role.

#### **Role** Chairperson

##### **Admission Constraints**

```

#(members(thisRole)) = 0
& room.isPresent(thisUser)
& member(thisUser,Manager)

```

Following is an example of role activation constraints in the *Secretary* role in the *Meeting* activity. Based on the specified constraints, a member of the *Secretary* role can perform operations only when at least one member of each of the *Manager* and the *Accountant* roles are present in the *room*.

#### **Role** Secretary

##### **Activation Constraints**

```

room.isAnyPresent(members(Manager))
& room.isAnyPresent(members(Accountant))

```

### 3.5. Role operation specification

Members of a role can perform a role operation only when the operation precondition, if any, is satisfied. The syntax of the role operation is presented in Fig. 8. As part of a role operation methods are invoked on objects. The action part of an operation may include invocation of methods on shared or private objects in the collaboration space.

In the *Meeting* activity example, a person in the *Accountant* role can perform the *DisplayFinancialData* operation. In this activity a context-based security policy requires that the accountant is allowed to display financial data only when three constraints are satisfied: (1) the *Chairperson* role must have executed the operation *ApprovePresentation*, (2) the member of the *Accountant* role who performs the operation has to be present in the

---

```

Reaction reactionName
  When {Event}
  [Precondition Condition]
  [Action {objId methodSignature methodParameter}]

```

---

Fig. 9. Reaction syntax.

meeting room, and (3) the person in the *Chairperson* role must also be present in the room. The operation specification with these preconditions is shown below.

```

Role Accountant
  Operation DisplayFinancialData {
    Precondition
      #(Chairperson.ApprovePresentation) = 1
      & room.isPresent(thisUser)
      & room.isPresent(member(Chairperson))
    Action projector.display(data)
  }

```

### 3.6. Reaction specification

Construction of smart environments requires the implementation of services that automatically execute some actions when certain events occur. Such system services are modeled as activities having no roles, and the automatic operation invocation is modeled as a *reaction*.

In contrast to an operation, a reaction is not invoked by a user but is automatically executed when certain events occur. Similar to an operation, a reaction is executed only when its precondition is true. As shown in Fig. 9, a reaction specification consists of three parts. The reaction is invoked when any of the events specified in the *When* clause occurs. If the precondition is true, then the corresponding action is executed. Reactions can be specified in the context of an activity or a role.

Fig. 10 shows an example of reactions in a room controller activity. When this activity is instantiated, it is given a specific room, which contains a projector and a light controller. Within the *RoomController* activity three reactions are defined which, respectively, switch on the light, switch off the light, and dim the light depending on certain specified conditions. The preconditions for switching on or off the light in the room are based upon the number of users in the room. Also, the lights in the room are dimmed if the projector is powered on. These reactions are invoked when the room object delivers an event of type *statusChangeEvent*.

### 3.7. Binding specification

Binding of a name in an activity's namespace to a resource is specified at different levels in our specification model. The binding primitive has the form shown in Fig. 11.

The name *objId* represents a resource or service. The optional tag *final* specifies that the binding is final and will not change. The tag *When* is used to specify context-based

```

Activity RoomController
  Object room ..
  Reaction SwitchOnLight
    When room.statusChangeEvent
    Precondition #(room.presentUsers()) > 0
    Action room.light.switchOn()
  Reaction SwitchOffLight
    When room.statusChangeEvent
    Precondition #(room.presentUsers()) = 0
    Action room.light.switchOff()
  Reaction DimLight
    When room.statusChangeEvent
    Precondition #(room.presentUsers()) > 0
    & room.projector.checkOnStatus()
    Action room.light.setLevel(LOW)

```

Fig. 10. Reaction specification in a room controller activity.

---

```

Bind objId [final]
  [When {Event}]
  With [new CodeBase | direct URL | objId
    | discover ({attribute=value}, ANY | ALL)]

```

---

Fig. 11. Object binding syntax.

binding requirements. It is followed by a list of events that would cause the specified binding directive to be executed. These events can be internal, i.e. generated within the activity, or imported from the environmental objects. The *When* clause is optional. Thus, an activity can have binding directives that do not depend on any events. Such binding directives are executed at the time of activity creation.

The resource binding policies can be specified in four different ways:

1. *Binding to a new object*: The binding primitive with *new* specifies that the resource should be created and should be bound to the name *objId*. Codebase specifies the type of the resource. For example, in a meeting activity, a whiteboard object is created and bound to the name *whiteboard* as:

```
Bind whiteboard final With new(//codeBase/WhiteBoard)
```

2. *Binding to an existing resource through URL*: This form of binding primitive with *direct* specifies that the resource identified by the given URL should be used in binding. This binding primitive is used when the resource already exists and its location is known. For example, within an activity the URL of the location service might be well known.

```
Object locationService RDD LocationServiceRDD
Bind locationService With direct (//LocationServiceURL)
```

3. *Binding to another object*: This form of binding primitive specifies that the name *objId* should be bound to the resource or service that is currently being referenced by

another name. A resource created as a part of a role can be exported to the activity scope by specifying that a name in the activity scope should be bound to a resource which has been created in the scope of the role. For example, consider a paper review activity containing the *Reviewer* role. Members of the *Reviewer* role are allowed to create private *review* objects. Once a review is done, the reviewer puts the private *review* object in the activity scope, by adding it into a collection called *submittedReviews* in the activity namespace. This is done as follows:

**Bind** review **With new**(//codeBase/Review)

.....

**Bind** submittedReviews **With** review

The collection called *submittedReviews* contains a set of objects, which are the reviews submitted by the reviewers. Here the binding operation adds a *review* object into this collection.

4. *Binding through discovery*: This form of the binding primitive is useful when a resource with a particular set of attributes is needed to be discovered in the environment. Each object name in the activity is considered to be of a particular RDD type. In the binding directive, some of the attributes of a required resource may need to be determined at the time of discovery, based on certain context information. Thus an RDD can be viewed as parameterized with certain attribute values being filled in based on the context. Consider an activity that wants to monitor the movement of a specified user. This activity subscribes to *locationChange* events from the location service. Whenever this event is received, the binding directive for the *camera* object is executed to discover and bind a camera at the specified user's current location. In the following example specification, members of the *SecurityGuard* role can monitor the user by invoking the *MonitorUser* operation, which plays the video-stream from the currently bound camera.

**Activity** UserTracking

**Parameter** userName

**Object** locationService **RDD** LocationServiceRDD

**Bind** locationService **With direct** (//LocationServiceURL)

**Object** camera **RDD** CameraRDD

**Bind** camera **When** *locationService.locationChange(userName)*

**With discover**(*location=locationService.getUserLocation(userName)*)

**Role** SecurityGuard

**Operation** MonitorUser

**Action** camera.playVideoStream()

#### 4. Examples of programming context-based applications

In this section we present several examples of programming pervasive computing applications using the specification model presented in the previous section. These examples illustrate the specification of various context-based resource access and security policies.

**Activity** Distributed Meeting

**Object** locationService **RDD** Location-Service-Description

**Object Collection** meetingRooms **RDD** Required-Room-Description

**Object Collection** projectorGroup **RDD** Required-Projector-Description

**Bind** locationService **With direct**(//LocationServiceURL)

**Bind** meetingRooms **With direct** (//Room1URL)

**Bind** meetingRooms **With direct** (//Room2URL)

**Bind** projectorGroup **With** meetingRooms.projector

**Role** Accountant

**Object** myRoom **RDD** Required-Room

**Object** myProjector **RDD** Required-Projector

**Bind** myRoom **When** locationService.locationChange(thisUser)  
**With** meetingRooms.select(isPresent(thisUser))

**Bind** myProjector **When** locationService.locationChange(thisUser)  
**With** myRoom.projector

**AdmissionConstraint**  
members(thisRole) < 1

**ActivationConstraint**  
myRoom.isPresent(members(Chairperson))  
& myRoom.isPresent(thisUser)

**Operation** DisplayData  
**Action** myProjector.displayData()

**Role** Chairperson

**AdmissionConstraint**  
#(members(thisRole)) < 1

**Operation** DisplayData  
**Action** projectorGroup.displayData()

Fig. 12. Distributed meeting activity specification.

#### 4.1. Distributed meeting specification

Consider a distributed meeting occurring in two meeting room locations. There are four roles in the meeting, viz. *Chairperson*, *Accountant*, *Secretary* and *Participant*. Members of each role can be present in any room. One particular room is reserved for discussing confidential matters such as financial data whereas the other room is open for more public discussions. Following are some of the context-based requirements for such a distributed meeting.

- R1 An operation in the *Accountant* role can only be performed when a member of the *Chairperson* role is also present in the same room where the accountant invoking the role operation is located.
- R2 Presentations performed by a person in the *Accountant* role should be displayed only in the room in which that accountant is present.
- R3 Presentations performed by the *Chairperson* role should be displayed in both the rooms.

Fig. 12 shows a partial specification of the distributed meeting activity with two roles: *Accountant* and *Chairperson*. There are two collection objects defined in the activity, one

```

Activity Museum Infodesk
Object locationService RDD Location-Service-Description
Parameter userPreference
Bind locationService With direct(//LocationServiceURL)
Role Visitor
Object audioDevice RDD Audio-Device-Description
Bind audioDevice When locationService.locationChange(thisUser)
With discover( <location=locationService.getLocation(thisUser),
                language=userPreference.preferredLanguage>)

```

Fig. 13. Specification of a customized museum information activity.

for rooms and another for the projectors inside the rooms. Two room objects, specified by their URLs, are added to the *meetingRooms* collection using the bind primitive. The *projectorGroup* collection refers to the projectors inside the rooms in the *meetingRooms* collection. The *Accountant* role declares for each of its members a private namespace consisting of *myRoom* and *myProjector* objects. For a person in the *Accountant* role, *myRoom* object is bound to the room in which that person is currently present. This is accomplished by performing *select* operation on the *meetingRooms* collection based on a selector function *isPresent*. The select operator returns an element in the specified collection for which the selector function is true. The object name *myProjector* is bound with the projector based on the binding of the object *myRoom*.

In Fig. 12, the first requirement R1 is satisfied by specifying the activation constraint for the *Accountant* role, which ensures that the accountant and the person in the *Chairperson* role are both present in the same room. R2 is satisfied based on the private binding of the *myProjector* in the *Accountant* role. When the accountant moves from one room to another, the binding of the *myProjector* changes. This is achieved by specifying the execution of the binding directive for *myRoom* for an accountant to be triggered whenever the location service delivers a *locationChange* event for that person. For the last requirement R3, in the *Chairperson* role, the operation of displaying data is invoked on the *projectorGroup* collection. This ensures that the presentation by the *Chairperson* is seen in all the rooms.

#### 4.2. Customized user information activity

In Fig. 13, we present the specification of a museum information desk activity. In this activity, the audio device of the user needs to bind with the audio player based on the user's location and also taking into consideration the user's choice of the language. In Fig. 13, the *audioDevice* object is re-bound when there is a change in the user's location. The *discover* primitive used in binding of the *audioDevice* object specifies the location attribute and the preferred language in the *Audio-Device-Description* to be used during resource discovery. The required RDD is filled in with the user's current location and the user's choice of the language as set in the object *userPreference*.

#### 4.3. Patient monitoring in a hospital

In a patient monitoring activity, a nurse is allowed to access a patient's records only when located close to the patient and also when the nurse's computing device is close to

```

Activity Patient Health Monitoring {
.....
Role Nurse
Operation AccessPatientData
Precondition
    locationService.getLocation(thisUser)
        = locationService.getLocation(thisUserDevice)
    & locationService.getLocation(thisUser)
        = locationService.getLocation(patientID)
Action ...

```

Fig. 14. Specification of health-care monitoring activity.

the nurse. This requirement is expressed as shown in Fig. 14 in the precondition of the *AccessPatientData* operation.

Some devices might provide functionality for proximity detection. Assuming that the device object provides an abstraction for proximity sensor in the form of a boolean function *proximitySensor(userID)*, the above requirement can be expressed as shown below. In this specification, precondition queries device context object.

```

Operation AccessPatientData
Precondition thisUserDevice.proximitySensor(thisUser)
Action ...

```

## 5. Middleware components and runtime execution model

In our earlier work we developed a middleware to build runtime environments for collaboration systems from their specifications [5]. To support the context-based ubiquitous collaboration model presented here, we extended our earlier work on this middleware to support dynamic resource discovery and binding, and context-based security and coordination policies. We briefly outline in this section the core set of components of this middleware architecture.

This middleware is developed using facilities of the Ajanta mobile agent programming framework [6]. The Ajanta framework provides a naming and certification service and supports object/code migration. It also provides authenticated communication for Java-RMI based interactions among distributed components. For supporting resource discovery in pervasive computing environments, we developed an RDD-based resource discovery service. Services and resources are registered with this discovery service with their RDDs and URLs for the RMI interface. Another facility provided by this middleware framework is a secure and trusted environment for executing various policy management components for an activity. These components are described below.

To manage a collaboration activity, from its specification we derive policies related to different management aspects. These aspects are related to management of roles and objects. The policies for a role pertain to role membership management, enforcement of context-based security and coordination requirements, and subscription and notification of context-related events. For an object, the policies pertain to context-based resource

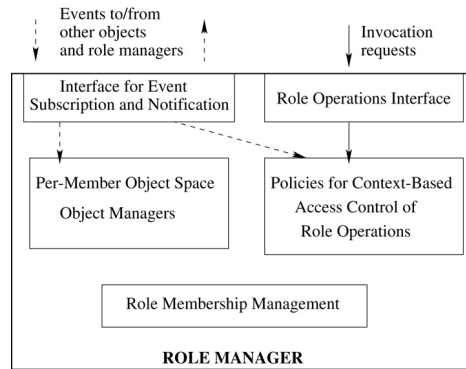


Fig. 15. Generic role manager.

discovery and binding, context-based access control requirements, and policies for subscription and notification of events.

To construct the runtime environment to manage an activity and enforce the policies specified for the activity, the middleware provides two types of generic managers: *role manager* and *object manager*. When an activity is instantiated, a role manager is created for each of the roles and an object manager for each of the objects defined within the activity. A specific manager for a role or an object is created from the corresponding generic manager by integrating with it the policies derived from the activity specification.

In Fig. 15, the components of a role manager are presented. A function of a role manager is to manage the user memberships in the role and enforce role admission policies. Moreover, it maintains an object-space for each member in the role. For each object in an object-space, the role manager maintains a references to the corresponding object manager. Role managers provide two remote interfaces: one is for role specific operations, and the other is for event subscription and notification. Event subscription policy specifies the subscribers for the operation-related events generated by this role. These events are required by other role or object managers for context-based policy enforcement. In contrast, event notification policy specifies the events that must be subscribed to by other managers to enforce preconditions for operations and role admission. The event notification policy is needed to ensure that events are received only from valid managers.

The components of an object manager are presented in Fig. 16. An object manager maintains a reference to a resource or service and enforces context-dependent object binding and access policies. It provides a remote interface for role managers to invoke methods on the object as part of role operations and reactions. Similar to a role manager, an object manager also has an event subscription and notification interface. Events notified to an object manager are used to trigger context-based resource binding and enforce dynamic access control policies. An object manager may also generate certain events representing specific changes in the state of the object. Such events are declared in the object's RDD specification, and other managers subscribe to these events according to the activity specification.

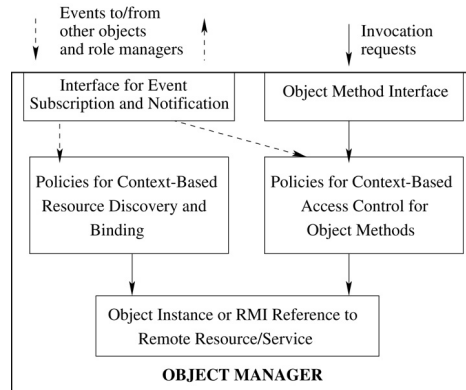


Fig. 16. Generic object manager.

In the interaction model supported in this environment, users can only interact with role managers. When a user invokes an operation to access an object, the role manager interacts with the corresponding object manager to initiate a *session* between the user and the object. Each user participates in the environment through a *User Coordination Interface (UCI)* executing on the user's computing device. The UCI maintains the RMI stubs for the various roles in which the user can be a member. Also, during any interactions with a role manager, it provides any device-specific certificates and context-information to the role manager.

Using the *Distributed Meeting* specification in Fig. 12, we illustrate here different policies that are integrated with generic managers to realize a collaborative activity. In the activity namespace of the *Distributed Meeting* two roles – *Accountant* and *Chairperson* – and three objects – *locationService*, *meetingRooms*, and *projectorGroup* – are defined. When a *Distributed Meeting* activity is instantiated, managers for these roles and objects are created. In the role-member namespace of the *Accountant* role, two objects – *myRoom* and *myProjector* – are defined for each member in the role, and the corresponding object managers are created.

We use the *Accountant* role and the *myProjector* object as examples to illustrate integration of different types of policies with generic managers based on the given specification. Two types of policies are derived and integrated with the *Accountant* role manager. First, the role admission control policy for this role specifies that at a given time there can be only one member in this role. Second, the context-based operation invocation policy specifies that the *DisplayData* operation can be invoked only when the member of the *Chairperson* role as well as the invoker of the operation are both present in the *myRoom* object. This policy module is created with a reference to the *Chairperson* role manager, which is queried for its member-list, and a reference to the *myRoom* object manager, which is queried to verify users' presence in the *myRoom* object.

For each member in the *Accountant* role, an object manager for *myProjector* is created with three types of policies. The first policy is for context-based resource binding, which specifies that the reference to the object has to be re-bound whenever the corresponding

member's location changes. The second is the event subscription-notification policy, which specifies that the *locationChange* event should be subscribed from the *locationService* object. This event is required for triggering the execution of context-based binding policy. The third policy is for access control, and it specifies that only a member of the *Accountant* role can invoke the *displayData* method of this object.

## 6. Related work

In the past several research groups have developed system architectures and programming models for building pervasive computing applications [7–9]. Most of these projects have used methods of custom design, implementation, and integration of services and components to realize a specific pervasive computing environment such as a smart home or a smart meeting room. There are two aspects in which our work differs from these research efforts. First, we deal with pervasive computing applications involving more than one user. Second, our approach consists of a specification-driven middleware to build pervasive computing applications having different kinds of policies, which may be based on the user's environmental context. Other projects which have taken this kind of specification-based approach are [10,11]. The trigger-based programming model [12] for identifying context changes is similar to our notion of trigger events. In our model both external environmental context events and internal activity events are used in the trigger conditions.

Specification of resource requirements based on contextual information for resource discovery in mobile and pervasive computing environments is addressed by others, such as [13,14]. In contrast, we present a model for collaboration among users in pervasive environments including resource requirements in the context of users and service interactions. The *pervasive activity* model, presented in this paper, includes specification of service descriptions, resource requirements, resource access policies, proactive actions by services in the context of activities, as well as coordination among collaborating users to access shared resources.

Middleware support for pervasive computing applications involving data staging architecture [15] and event heap abstraction [16] has been studied. Our middleware framework is driven by application level policy specification for context-based resource access and coordination among users and environmental services.

The concept of roles for specification of resource access policies in pervasive computing environments has been explored by others in the past [17,18]. The work presented in this paper is extended from our earlier work on a role-based specification model for collaboration systems [3]. Here we present an extended model for specification-driven construction of collaborative applications in pervasive computing environments. This model can express a rich set of context-based constraints on resource discovery, user interaction, and resource access.

## 7. Conclusions

We have presented here an approach for constructing collaborative activities in pervasive computing environments using high level specifications. This work extends

our previous work on a specification model for distributed collaboration systems [3] and development of a middleware to realize such systems [5]. The primary contribution of this paper is a specification model for collaborative activities that are immersed in pervasive computing environments. The model is developed to express context-based requirements for resource discovery, dynamic policies for secure resource access, and coordination among users and services in the environment. Using a set of examples, we have shown here the expressiveness of this model for specifying various kinds of context-based resource requirements and security policies.

## References

- [1] R. Sandhu, E. Coyne, H. Feinstein, C. Youman, Role-based access control models, *IEEE Computer* 29 (2) (1996) 38–47.
- [2] A.K. Dey, Understanding and using context, *Journal of Personal and Ubiquitous Computing* 5 (1) (2001) 4–7.
- [3] A. Tripathi, T. Ahmed, R. Kumar, Specification of secure distributed collaboration systems, in: *IEEE International Symposium on Autonomous Distributed Systems, ISADS, 2003*, pp. 149–156.
- [4] A. Tripathi, T. Ahmed, D. Kulkarni, R. Kumar, K. Kashiramka, Context-based secure resource access in pervasive computing environments, in: *1st IEEE International Workshop on Pervasive Computing and Communications Security, PerSec'04, 2004*, pp. 159–163.
- [5] A. Tripathi, T. Ahmed, R. Kumar, S. Jaman, Design of a policy-driven middleware for secure distributed collaboration, in: *Proceedings of the 22nd International Conference on Distributed Computing Systems, ICDCS, 2002*, pp. 393–400.
- [6] A. Tripathi, N. Karnik, T. Ahmed, R. Singh, A. Prakash, V. Kakani, M. Vora, M. Pathak, Design of the Ajanta system for mobile agent programming, *Journal of Systems and Software* 62 (2002) 123–140.
- [7] MIT Project Oxygen, Available at url <http://oxygen.lcs.mit.edu/>.
- [8] M. Satyanarayanan, Pervasive computing: vision and challenges, *IEEE Personal Communications* 8 (4) (2001) 10–17.
- [9] B. Schilit, N. Adams, R. Want, Context-aware computing applications, in: *IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, USA, 1994*, pp. 85–90.
- [10] C.K. Hess, M. Roman, R.H. Campbell, Building applications for ubiquitous computing environments, in: *International Conference on Pervasive Computing, 2002*.
- [11] S.S. Yau, F. Karim, Y. Wang, B. Wang, S.K. Gupta, Reconfigurable context-sensitive middleware for pervasive computing, *IEEE Pervasive Computing* 1 (3) (2002) 33–40.
- [12] K. Henriksen, J. Indulska, A software engineering framework for context-aware pervasive computing, in: *Second IEEE International Conference on Pervasive Computing and Communications, 2004*, pp. 77–86.
- [13] G. Chen, D. Kotz, Context-sensitive resource discovery, in: *First IEEE International Conference on Pervasive Computing and Communications, 2003*, pp. 243–252.
- [14] A. Cedilnik, L. Kagal, F. Perich, J. Undercoffer, A. Joshi, A secure infrastructure for service discovery and access in pervasive computing, Tech. Rep. TR-CS-01-12, Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, 2001.
- [15] D. Garlan, D. Siewiorek, A. Smailagic, P. Steenkiste, Project Aura: toward distraction-free pervasive computing, *IEEE Pervasive Computing* 1 (2) (2002) 22–31.
- [16] R. Ballagas, A. Szybalski, A. Fox, Patch Panel: enabling control-Flow interoperability in ubicomp environments, in: *Second IEEE International Conference on Pervasive Computing and Communications, 2004*, pp. 241–252.
- [17] W.K. Edwards, Policies and roles in collaborative applications, in: *Proc. of CSCW'96, 1996*, pp. 11–20.
- [18] M.J. Covington, W. Long, S. Srinivasan, A.K. Dey, M. Ahamad, G.D. Abowd, Securing context-aware applications using environment roles, in: *Symposium on Access Control Models and Technologies, 2001*, pp. 10–20.



**Anand Tripathi** received his M.S. and Ph.D. degrees in electrical engineering from the University of Texas at Austin, in 1978 and 1980, and B.Tech in electrical engineering from the Indian Institute of Technology, Bombay, in 1972. His research interests are in distributed systems, middleware architectures, collaboration systems, pervasive computing, system security, and fault-tolerant computing.

He is a professor of computer science at the University of Minnesota, Minneapolis. He worked as a Scientific Officer at Bhabha Atomic Research Center, India, during 1973–1975. During 1981–1984 he worked as a Senior Principal Research Scientist at Honeywell Computer Science Center, Minneapolis. He joined the University of Minnesota in 1984. During 1995–1997, he served as a Program Director in the Division of Computer and Communications Research at the National Science Foundation, Arlington, Virginia.

Prof. Tripathi is a member of IEEE Computer Society and ACM. Currently, he is serving as an at-large member of the IEEE Computer Society Publications Board (2001–2005), and member of the editorial boards of IEEE Transactions on Computers, IEEE Pervasive Computing, and IEEE Distributed Systems Online. He served as a Program Vice Chair for International Conference on Distributed Computing Systems (in 1997). He was the Program Chair for the IEEE Symposium on Reliable Distributed Systems in 2001. He served as the Program Chair for the Second IEEE International Conference on Pervasive Computing and Communications (PerCom) 2004, and a Vice Chair for this conference in 2003. He served as the Program Chair for the IEEE Workshop on Mobile Distributed Computing (MDC) held in June 2003. He was one of the organizers of two ECOOP (European Conference on Object Oriented Programming) Workshops on exception handling (2000 and 2003), and co-editor for a Springer LNCS volume on exception handling, published in 2002.



**Devdatta Kulkarni** is a Ph.D. student in the Department of Computer Science & Engineering at the University of Minnesota, Twin Cities. He received M.S. in Computer Science from the University of Minnesota, Duluth, in 2002. His research interests are in distributed systems, pervasive computing, and sensor networks.



**Tanvir Ahmed** received Ph.D. degree in computer science from the University of Minnesota, Twin Cities in 2004. His dissertation addresses the research area of security in CSCW systems. He has worked and published papers on research areas, such as, policy specification and verification, middleware for collaboration systems, mobile agents, network monitoring, autonomic computing, and security in pervasive computing. He has received the M.S. degree in computer science from the University of Minnesota in 1999, and the B.S. degree in computer science from the University of Mississippi in 1995.