Programming Secure and Robust Pervasive Computing Applications

Devdatta Kulkarni, Anand Tripathi, and Tanvir Ahmed
Department of Computer Science
University of Minnesota, Minneapolis MN 55455
Contact Author Email: tripathi@cs.umn.edu

Abstract

We have developed a programming framework for building context-aware multi-user collaborative applications in pervasive computing environments. It supports context-sensitive security and multi-user coordination requirements. It also supports error handling in pervasive computing applications through an exception handling model. In this paper we present the programming framework and demonstrate its utility for building context-aware, secure, fault-tolerant pervasive computing applications. We also demonstrate integration of an agent-based distributed event monitoring system for gathering context information that is derived from sensor data.

I. Introduction

In the last few years there has been considerable interest in the research community in building pervasive computing environments. Several research groups have developed system architectures and programming frameworks for building pervasive computing applications [1], [2], [3], [4], [5]. There is a large spectrum of pervasive computing applications reported in the literature [1], [2], [3], [4], [5], [6], [7]. An important characteristic of such applications is the use of context information to enrich user experience through adaptation and augmentation of their computing environment [8]. Such applications transparently provide access to required resources and data based on the contextual information. A user's physical location has been one of the most prominent context information used in many such applications, allowing mobile users to seamlessly perform their activities while moving across different physical spaces. A large body of research has addressed the issues related to modeling and representation of context information involving other aspects such as the characteristics of the user's computing environment (e.g., available bandwidth or devices), co-location with other users, and the user's activity related preferences and profiles [9], [10], [11].

Our work differs from the other research in this area in three aspects. First, our work addresses context-based security and coordination requirements of collaborative applications in pervasive computing environments. Most of the pervasive computing applications and the middleware environments reported in the literature do not address multi-user coordination and security issues. Though many of the applications reported in the literature involve multiple users, the requirements of security are addressed purely from the perspective of single user. Second, we provide a programming framework which allows one to realize pervasive computing applications from their high level specification by automated integration

This work was supported by National Science Foundation grant 0411961.

of application components, environmental services, and context sensors. Only few other projects [1], [5], have taken this kind of approach. The commonly used approach in other projects is to build such applications through custom integration of components and services. Third, our work provides error handling in pervasive computing applications by integrating an exception handling model in our programming framework. The error handling issues in such environments have not been adequately addressed by the research community.

We view a pervasive computing environment as a collaboration space involving multiple users and environmental agents that collaborate for certain application-defined objectives. This approach stems from our earlier work on building secure distributed collaboration environments from their high level specification [12], [13]. The conceptual model of this approach was presented in [14], [15], where we extended our earlier work to support context-based resource discovery/binding and context-based secure coordination among users and environmental agents in pervasive computing applications.

In this paper we present the realization of this model in a programming framework and illustrate its capabilities for supporting context-based security, dynamic resource binding, and exception handling through the development of a comprehensive case study application. This is a collaborative application involving multiple users in a secure distributed meeting environment that spans multiple physical locations. This application is intended to support context-sensitive security requirements that depend on users' presence in some physical space and their co-location with other users. We also present integration of an exception handling model for pervasive computing applications that we had proposed [16], in our programming framework. Lastly, we illustrate how an agent-based distributed event monitoring system [17], [18], is used for gathering and providing context information.

Our work addresses important requirements related to four aspects of context-aware collaborative applications in pervasive computing environments: context-based security and coordination, context-aware proactive actions, context-based dynamic discovery and binding of resources , and exception handling for robust operations. We consider context from an application's perspective as *internal context* or *external context*. The application's execution state, such as the tasks performed by each user in a multi-user application, forms its *internal context*, while any information that can be sensed from the external world, e.g. a user's physical location, forms its *external context*.

Context-based security and coordination: Central to our programming framework is a role-based security model. A role represents permissions for users to perform actions in a collaborative application. We represent a role's permissions through a set of operations associated with the role. Dynamic security and coordination requirements of a role-based collaborative application requires that the user's privileges to execute role operations in such applications be based on certain internal or external context conditions.

Context-aware Proactive Actions: Pervasive computing applications need mechanisms for performing proactive actions based on the context information. Examples of such proactive actions include turning off lights in a room when no user is present in the room or transporting user's computing environment to the device close to the user. In our programming framework such proactive actions are supported through the *reaction* construct.

Context-based dynamic resource discovery and binding: An application defines a name-space for the resources required in its execution. The binding of names in a name-space with physical resources in an environment may need to be changed based on the context of the application. In our programming framework we support different kinds of directives for resource discovery and binding.

Exception handling for robust operations: One of the characterizing features of pervasive computing applications is the dynamically changing integration of application components and environmental services/resources. Different types of application errors are highly probable in such environments, such as: required resources may not be always available in the application's current environmental context, the context condition under which a secure resource usage is permitted may get violated because of sudden changes in the environmental context, or a role member may encounter errors while interacting with a resource/service. We handle such errors through the exception handling model integrated in our programming framework.

Section II presents the distributed meeting application and its context-based requirements. Section III presents the basic elements of our programming framework. In Section IV we present the complete specification of the distributed meeting application. In Section V we present the implementation details of this application. In Section VI we present the error conditions that arise in it and demonstrate how they are handled in our programming framework through the exception handling model. Section VII discusses the related work, and the conclusions are presented in Section VIII.

II. Context-Sensitive Secure Distributed Meeting

Consider a distributed meeting involving participants present in multiple rooms. Each meeting room consists of a projector attached to a display device. The meeting application creates two audio channel objects which can be connected to the users' input devices such as a microphone and output devices such as audio players. A user participates in the meeting using his/her own laptop/PDA with a microphone and an audio player. The meeting may involve discussions related to classified as well as unclassified information. The two audio channels, called *private channel* and *public channel*, are designated to be used separately for classified and unclassified presentations.

In this distributed meeting there are three roles corresponding to *Chairperson*, *Participant* and *Secretary*. The *Chairperson* and the *Secretary* role each have only one member, and all other authorized users are in the *Participant* role. The *Participant* role is provided with operations through which its members can perform presentations of the classified or unclassified information. Such a distributed meeting has the following context-sensitive security and resource access requirements:

R1: The presentation of the classified information by a user in the *Participant* role should only be directed to the projector of the room in which that user is located and the audio presentation should be directed to the *private channel*.

R2: The presentation of the unclassified information has no such restriction. It should be directed to projectors in all the meeting rooms and both the audio channels.

R3: A user in the *Participant* role can perform a classified presentation only when co-located with the *Chairperson* role member in the same room. Furthermore, the *Chairperson* has to explicitly grant permission to that user for making the classified presentation.

R4: The room in which the user making a classified presentation and the chairperson are present becomes the room for confidential discussions. Only those participants who are present in that room can talk on and listen to the private audio channel. They cannot talk on the public audio channel but they can listen to the public channel. On the other hand, participants in other rooms can talk and listen only on the public audio channel.

R5: Some participants and the chairperson may move from one room to another. Requirement R4 above should be satisfied corresponding to the room in which the chairperson is currently present.

R6: The projector in a room should be enabled if and only if there is any authorized user present in that room.

Corresponding to the above application requirements the following functionalities need to be supported in the programming framework.

P1: We need mechanisms for binding resources based on the context information. For example, for making classified presentations as part of requirement R1, each *Participant* role member needs to have access to the projector in the room where he/she is currently located, and for requirement R5 this resource binding needs to change if the participant moves from one room to another. This resource binding depends on the context information related to a particular role member and it is completely independent of other role members' context. Each such resource has to be identified with a name that is private to a role member. On the other hand, global resource names are needed to refer to resources that are shared among all the roles, for example, public and private audio channels in this application. This highlights two kinds of requirements: one is the separation of global and role member's private object name-spaces, and the other is the context-based dynamic binding of these names to resources.

P2: We need mechanisms for handling collection of resources of a particular type as a single entity. For example, as part of requirement R2, presentations related to unclassified data need to be directed to the projectors of all the meeting rooms. An action performed on the collection will be performed on all the objects in the collection.

P3: We need mechanisms for accessing application's internal and external context information. For requirement R3 we need internal context information corresponding to the execution of the operation for approving participant's presentation by the *Chairperson* role, and we need external context information corresponding to the location of the *Participant* role member and his/her co-location with the *Chairperson* role member in a particular room.

P4: We need mechanisms for constraining role operation invocation subject to external context information. For example, corresponding to requirement R3 we need to disallow presentations of the classified information by a user in the *Participant* role and corresponding to R4 we need to disallow participant's interaction with the private audio channel, if that user is not present in the same room where the *Chairperson* role member is located.

P5: We need mechanisms for performing proactive actions in the environment based on the context information. The need for such proactive actions is observed in requirement R6.

III. Conceptual Model of the Specification based Programming Framework

We present here the basic elements of the specification model. The specific constructs are presented in the next section as part of the distributed meeting example. In our programming framework, a pervasive computing application spanning over the Internet and involving distributed and mobile users is modeled as an *activity*. An *activity* represents a shared workspace for interactions among users and environmental agents. An activity definition contains three elements: a set of roles, an object name-space containing the definitions of the required shared objects and resources, and a set of context-driven proactive actions, called *reactions*.

A. Object Description and Binding

An object in an activity can refer to one of the three kinds of entities: a shared resource created by the activity, a system-level resource or service existing in the environment, or an object which provides abstractions for the physical environment. An object name can be defined either in the global scope, which is visible to all the members of all the roles, or it may belong to a particular role's *member-private name-space*. For the latter kind of name-space, a separate instance of this object space is maintained for each role member. An object name may refer to an *individual* resource instance or a *collection* of a similar type of resources. A collection represents a group of similar resources and any operation performed on a collection is executed on each member in the group.

Associated with each object name there is a binding specification, which identifies the resource to which that name should be bound to. Additionally, it also defines when the binding action should be executed. The binding actions may be performed during the initialization of the activity or may be *triggered* on the occurrence of external context events. There are three ways to specify the resource to be used in binding an object name: one is to create a new instance of a specified type and bind object name to it, second is to specify a resource to be bound through its URL, and the third is to discover a resource in the environment based on its description and bind the object name to it.

For resource description, we have developed an XML schema termed Resource Description Definition (RDD) [19] which combines the features of RDF (Resource Description Framework) and WSDL (Web Service Definition Language). A RDD description consists of attribute-value pairs, interfaces supported by the resource, and events exported by the resource.

B. Role Operations

A role operation has two parts: *precondition* and *action*. A role operation's precondition must be satisfied before the operation's action can be executed. The operation action starts a *resource access session* as part of which a set of methods may be invoked on an object. Internal and external context events and state of environmental objects are used as part of operation preconditions for enforcing context-based coordination and security requirements. Internal context events are related to the role operation invocations. Two types of events, *start* and *finish*, are defined for a role operation and are generated by the middleware framework indicating the beginning and the completion of a role operation session.

C. Reactions

A reaction represents a proactive action that is automatically executed when certain external context related events occur. A reaction is triggered by external context events and consists of preconditions and actions similar to a role operation. On triggering, the precondition of a reaction is evaluated, and the corresponding action is executed only if the precondition is true.

IV. Specification of Context-Sensitive Secure Distributed Meeting

Figure 1 shows the logical view of the distributed meeting activity that spans two rooms, and Figure 2 shows its XML specification template.

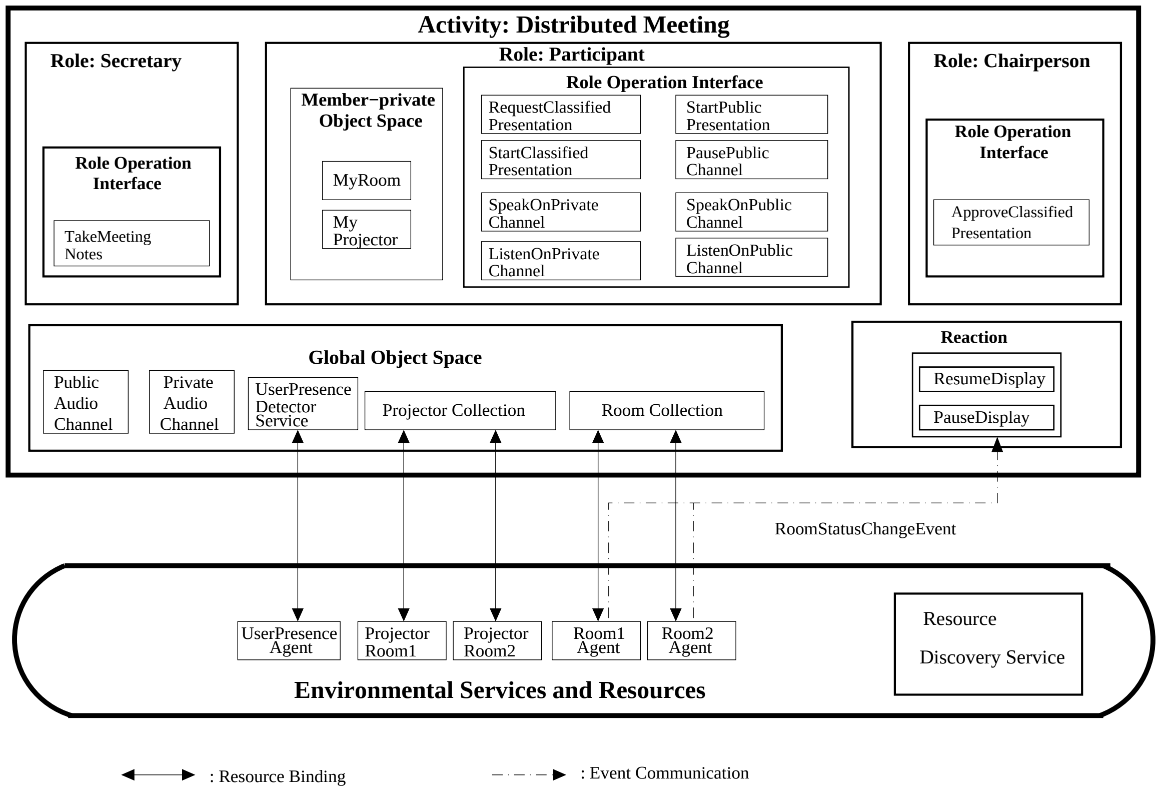

Fig. 1. Logical view of context-sensitive secure distributed meeting

A. Global Object Space and Object Binding

The meeting activity requires various different kinds of resources, and the corresponding objects are defined in the activity's global object space. Object names corresponding to public and private audio channels are bound with newly created audio channel objects. Two collections, *RoomCollection* and *ProjectorCollection*, are defined for the two meeting rooms and the two projectors in these two rooms. These are bound, using URLs, with the agents representing the two rooms and the two projector objects, respectively. An object *UserPresenceDetectorService* is bound with the *UserPresenceAgent* available in the environment, which provides location information for role members.

In our programming model, objects referring to an individual resource are defined using the *OBJECT_DEFINITION* tag, and objects referring to collection of resources are defined using the *OBJECT_COLLECTION* tag. Events may be imported from objects. Each such event is specified through the *IMPORT_EVENT* tag which is nested inside the *OBJECT_DEFINITION* tag or the *OBJECT_COLLECTION* tag. The meeting activity imports *UserArrivalEvent* from the *UserPresenceDetectorService*, and *RoomStatusChangeEvent* from the *RoomCollection*. Definition of *UserPresenceDetectorService* and *RoomCollection* is shown in Figure 3 and Figure 4, respectively. Binding of an object name with a resource is specified using the *BIND* tag.

B. Role Member-private Object Space and Object Binding

Two objects, *MyRoom* and *MyProjector* are defined in the member-private object space of each member in the *Participant* role. For each role member, *MyRoom* object refers to the room in which that user is currently present, and *MyProjector* refers to the projector in that room.

```
<ACTIVITY NAME=“DistributedMeeting”>
    <OBJECT_DEFINITION NAME=“PublicAudioChannel”> ..... </OBJECT_DEFINITION>
    <OBJECT_DEFINITION NAME=“PrivateAudioChannel”> ..... </OBJECT_DEFINITION>
    <OBJECT_DEFINITION NAME=“UserPresenceDetectorService”> ..... </OBJECT_DEFINITION>
    <OBJECT_COLLECTION NAME=“ProjectorCollection”> ..... </OBJECT_COLLECTION>
    <OBJECT_COLLECTION NAME=“RoomCollection”> ..... </OBJECT_COLLECTION>
    <ROLE NAME=“Secretary”>
        <OPERATION NAME=“TakeMeetingNotes”> ..... </OPERATION>
    </ROLE>
    <ROLE NAME=“Participant”>
        <OBJECT_DEFINITION NAME=“MyRoom”> ..... </OBJECT_DEFINITION>
        <OBJECT_DEFINITION NAME=“MyProjector”/>
        <OPERATION NAME=“RequestClassifiedPresentation”> ..... </OPERATION>
        <OPERATION NAME=“StartClassifiedPresentation”> ..... </OPERATION>
        <OPERATION NAME=“SpeakOnPrivateChannel”> ..... </OPERATION>
        <OPERATION NAME=“ListenOnPrivateChannel”> ..... </OPERATION>
        <OPERATION NAME=“StartPublicPresentation”> ..... </OPERATION>
        <OPERATION NAME=“SpeakOnPublicChannel”> ..... </OPERATION>
        <OPERATION NAME=“ListenOnPublicChannel”> ..... </OPERATION>
        <OPERATION NAME=“PausePublicChannel”> ..... </OPERATION>
    </ROLE>
    <ROLE NAME=“Chairperson”>
        <OPERATION NAME=“ApproveClassifiedPresentation”> ..... </OPERATION>
    </ROLE>
    <REACTION NAME=“PauseDisplay”> ...</REACTION>
    <REACTION NAME=“ResumeDisplay”> ...</REACTION>
</ACTIVITY>
```

Fig. 2. *DistributedMeeting*: Activity specification template

```
<OBJECT_DEFINITION NAME=“UserPresenceDetectorService”>
    <IMPORT_EVENT NAME=“UserArrivalEvent”/>
</OBJECT_DEFINITION>
<BIND OBJECT_REF=“UserPresenceDetectorService”>
    <DIRECT URL=“//UserPresenceAgent”/>
</BIND>
```

Fig. 3. *UserPresenceDetectorService*: Object definition and binding

Dynamic Object Binding: Discovery based binding primitive is used for binding these objects. The binding specification for the *MyRoom* object is shown in Figure 5. The discovery service uses the *RDD* of an object to search for an appropriate resource in its database. The *RDD* for the room is shown in Figure 6, with the attribute *LOCATION* declared as a parameter.

The *UserArrivalEvent* triggers this binding directive. The event contains the identity of the user for whom the event is generated. The attribute *LOCATION* in the *RDD* is filled by plugging the value obtained by invoking *getLocation* method on this event. The variable *thisUser* is passed as a parameter to this method. In our specification framework, the pseudo variable *thisUser* refers to the particular role member for whom an operation or a binding directive is executed. If the user name passed as the parameter to this method matches the user name inside the event then the symbolic name of the room in which the user is present is returned, otherwise null is returned. The binding of *MyProjector* is similar.

C. Role Operations

The *Participant* role has two sets of operations to perform presentations of classified and unclassified information. The operations *RequestClassifiedPresentation*, *StartClassifiedPresentation*, *SpeakOnPrivateChannel*, and *ListenOnPrivateChannel* are to be used for classified presentations, and the operations

```
<OBJECT_COLLECTION NAME="RoomCollection">
    <IMPORT_EVENT NAME="RoomStatusChangeEvent">
</OBJECT_COLLECTION >
<BIND OBJECT_REF="RoomCollection">
    <DIRECT URL="//Room1URL"/>
    <DIRECT URL="//Room2URL"/>
</BIND>
```

Fig. 4. *RoomCollection*: Object definition and binding

```
<BIND OBJECT_REF="MyRoom">
    <WHEN EVENT_NAME="UserArrivalEvent" FROM="UserPresenceDetectorService"/>
    <DISCOVER RDD_FILE="//Required-RoomRDD.xml">
        <SET_PARAM NAME="LOCATION">
            <METHOD_INVOCATION OBJECT_REF="UserArrivalEvent" METHOD_NAME="getLocation"
                ARG="thisUser"/>
        </SET_PARAM>
    </DISCOVER>
</BIND>
```

Fig. 5. *MyRoom* object binding

StartPublicPresentation, *SpeakOnPublicChannel*, *PausePublicChannel*, and *ListenOnPublicChannel* are to be used for unclassified presentations.

The pseudocode of the *StartClassifiedPresentation* is shown in Figure 7. For readability purpose we present here its specification in a pseudocode form instead of the XML specification[1]. In the pseudocode, the terms in **boldface** represent XML tags and # represents a count operator. The count operator, when applied to an event, returns the count of the times that event has occurred. The operation *StartClassifiedPresentation* starts a resource access session with the projector referred to by *MyProjector* and as part of this session various presentation related methods can be invoked on the projector. In order to satisfy requirements R3 and R4 we need that the *StartClassifiedPresenation* be allowed to be invoked only under the following conditions: (i) both the *Chairperson* role member and the *Participant* member invoking this operation are present in the same room; (ii) *Chairperson* has approved the presentation by executing the *ApprovePresentation* operation; (iii) the *Participant* role member's microphone is not connected to the *PublicAudioChannel*. These conditions are specified as part of the operation's precondition in Figure 7.

The first condition is evaluated by querying the presence of the participant and the chairperson by invoking method *isPresent* on the *MyRoom* object (line 3). The *Participant* role member's ID and the *Chairperson* role member's ID are passed as parameters to this method by specifying *thisUser* and *members(Chairperson)*, respectively. The function `members(RoleName)` returns the list of members present in the role specified by *RoleName*. The second condition specified in lines 4-5. The condition in line 4 checks for an outstanding request by the *Participant* role member. The condition in line 5 checks for the equality between the counts of *ApproveClassifiedPresentation* operation invoked by the *Chairperson* role member and the *RequestClassifiedPresentation* operation invoked by the *Participant* role member where the *grantee* and the *invoker* attributes in the events associated with these two operations are set to *thisUser*, respectively. This ensures that only that participant, for whom the *Chairperson* has granted approval, is able to execute this operation. The third condition specified in line 6, checks whether the difference in the counts of *SpeakOnPublicChannel* and *PausePublicChannel* operations invoked by

[1]XML specification of the activity is available at http://www.cs.umn.edu/Ajanta/publications.html/DistributedMeeting.xml

```
<RDD CATEGORY="ROOM">
    <INTERFACE>
        <OPERATION NAME="isPresent">
            <INPUTPARAM TYPE="String"/>
            <OUTPUTPARAM TYPE="boolean"/>
        </OPERATION>
        <OPERATION NAME="isPresent">
            <INPUTPARAM TYPE="Vector"/>
            <OUTPUTPARAM TYPE="boolean"/>
        </OPERATION>
        <OPERATION NAME="presentUserCount">
            <OUTPUTPARAM TYPE="Integer"/>
        </OPERATION>
    </INTERFACE>
    <ATTRIBUTELIST>
        <ATTRIBUTE NAME="LOCATION" VALUE="?PARAM"/>
        <ATTRIBUTE NAME="CAPACITY" VALUE="25"/>
    </ATTRIBUTELIST>
    <EXPORTEVENTS>
        <EVENT NAME="RoomStatusChangeEvent"/>
    </EXPORTEVENTS>
</RDD>
```

Fig. 6. *Required-RoomRDD.xml*: Parameterized Room RDD

thisUser is equal to zero, implying that currently the participant is not connected to the public audio channel. As part of the operation's action, method *display* is invoked on the *MyProjector* object (line 7).

The operation *SpeakOnPrivateChannel* connects the microphone associated with the *Participant's* device with the *PrivateAudioChannel* object. In Figure 8, the pseudocode for the *SpeakOnPrivateChannel* operation is shown. Its precondition is similar to the first condition of *StartClassifiedPresentation* operation precondition. Other *Participant* role operations are specified in a similar manner.

```
1.Role Participant
2.  Operation StartClassifiedPresentation
3.      Precondition MyRoom.isPresent(thisUser) & MyRoom.isPresent(members(Chairperson))
4.              & (#Participant.RequestClassifiedPresentation(invoker=thisUser) > 0)
5.          & (#Chairperson.ApproveClassifiedPresentation(grantee=thisUser) -
              #Participant.RequestClassifiedPresentation(invoker=thisUser) = 0)
6.          & (#Participant.SpeakOnPublicChannel(invoker=thisUser) - #Participant.PausePublicChannel(invoker=thisUser) = 0)
7.      Action MyProjector.display(//ClassifiedInformation)
```

Fig. 7. Participant role: *StartClassifiedPresentation* operation pseudocode

```
1.  Role Participant
2.      Operation SpeakOnPrivateChannel
3.          Precondition MyRoom.isPresent(thisUser) & MyRoom.isPresent(members(Chairperson))
4.          Action PrivateAudioChannel.registerToSend(thisUser)
```

Fig. 8. Participant role: *SpeakOnPrivateChannel* operation pseudocode

D. Reactions

Pseudocode of the reaction that pauses the projector is shown in Figure 9. The reaction is triggered by the *RoomStatusChangeEvent* received from any one of the room agents. The room agent corresponding to which the *RoomStatusChangeEvent* occurred is selected from the *RoomCollection* by matching the room agent's identifier with the location identifier in the *RoomStatusChangeEvent*, obtained by invoking the *getEventLocationID* method on this event. The number of users present in that room is queried by invoking *presentUserCount* method on the selected room agent (line 3). The projector belonging to that room is selected from the *ProjectorCollection* similar to the selection of the room agent as discussed above, and the presentation is paused by invoking *pauseDisplay* method on the selected projector (line 4), if the number of users in the room is zero.

```
1.Reaction PauseDisplay
2.  When RoomStatusChangeEvent
3.  Precondition (RoomCollection.select(RoomStatusChangeEvent.getEventLocationID())).presentUserCount() = 0
4.  Action (ProjectorCollection.select(RoomStatusChangeEvent.getEventLocationID())).pauseDisplay()
```

Fig. 9. *PauseDisplay* reaction pseudocode

V. Implementation of Context-Sensitive Secure Distributed Meeting

In order to realize an application in our programming framework we need to integrate three entities: middleware components that are derived from the activity specification by integrating policies with generic middleware components, context information agents, and environmental resources and services such as a discovery service.

A. Middleware Environment and Components

The runtime environment for a given application is generated by the middleware by constructing appropriate managers for the activities, roles, and objects defined in that application. For each such entity defined in the specification, a manager object is created by integrating a generic manager of that type with the policy modules that are derived from the activity specification. The middleware provides these generic managers [12], [13], [20]. All managers are run on a set of *trusted servers*, which provide a secure execution environment for policy enforcement.

The policy modules contain the policies related to event subscription/notification between various roles, policies for binding of object names to resources, and policies for context-based access control and role operation execution. Role managers control execution of role operations and generate the corresponding *start* and *finish* events. All resources are accessed through their corresponding object managers which provide method level access control through method tickets for all the method invocations on the resource.

Users are provided a *User Coordination Interface (UCI)* through which they can invoke the role operations. The GUI objects for resource access session started by a role operation are executed by the UCI.

B. Context Information Agents

Contextual information is gathered through sensors that are interfaced with a set of agents that generate events corresponding to the sensed environmental conditions. We have previously developed a middleware framework for agent-based distributed event monitoring system [17], [18]. Central to this framework are a set of event detectors and event handlers deployed in an Ajanta agent [21]. An event detector for a particular event type monitors some specific condition by sensing the physical environment and generates an event of that type. Such events may trigger other detectors in the same or remote agents. An event handler performs the function of triggering local detectors, storing the events in local or global databases, and sending them to remote subscribers.

Users participate in the meeting using their personal devices such as laptops and PDA that are enabled with Bluetooth. These devices are used for detecting user presence in the meeting rooms. Figure 10 shows the configuration of agents for our experiment involving two rooms. Each room is equipped with an agent that runs the *BluetoothEventDetector* which periodically performs Bluetooth discovery and generates the *BluetoothEvent* for each Bluetooth device discovered in the vicinity. This event contains the Bluetooth reader ID corresponding to the agent that detected the Bluetooth device and the detected Bluetooth device ID. This event triggers *DeviceLocationChangeEventDetector* that determines whether a Bluetooth device has *arrived in* or *departed from* the room by checking its membership in the Bluetooth device list that was obtained in the previous discovery cycle. This event then triggers *RoomStatusChangeEventDetector* which uses a database of known reader locations to determine the location of the reader that detected this Bluetooth device.

The *RoomStatusChangeEvent* from both the agents is subscribed by the *UserPresenceAgent*. This agent contains three detectors: a *UserStatusChangeEventDetector*, a *UserArrivalEventDetector*, and a *UserDepartureEventDetector*. The *UserStatusChangeEventDetector* uses a database to map the Bluetooth device ID to a particular user name. A *UserStatusChangeEvent* is generated only if the Bluetooth device ID maps to a valid user name known in the system. The *UserStatusChangeEvent* locally triggers the *UserArrivalEventDetector* and the *UserDepartureEventDetector*. These detectors respectively generate *UserArrivalEvent* and *UserDepartureEvent* depending on the status of the Bluetooth device, set to *arrival* or *departure*, inside the *UserStatusChangeEvent*.

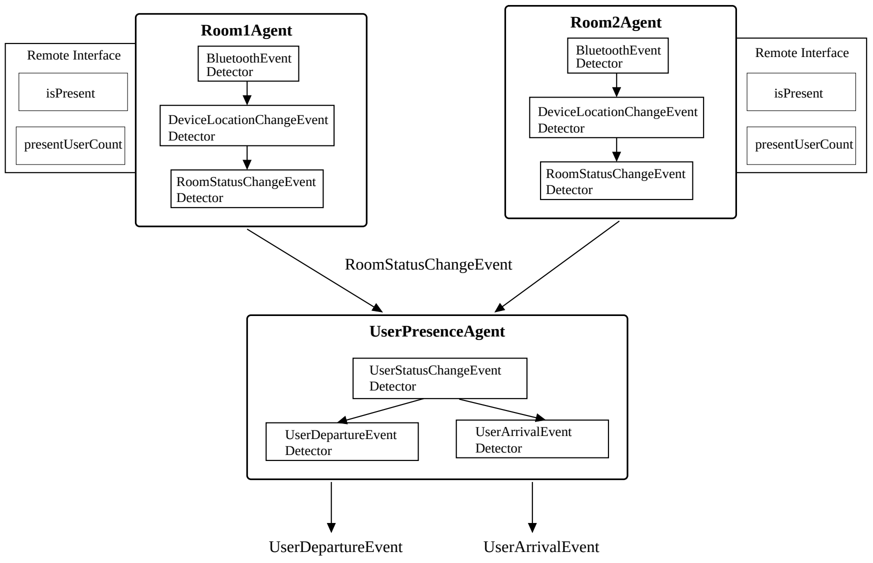

Fig. 10. Configuration of Context Information Agents

C. Discovery Service

A discovery service is provided in the middleware for discovering and binding required resources in the environment. A resource/service registers the *RDD* of the resource, the interface definition, and the Java RMI URL of the resource/service with the discovery service. The discovery service is queried by object managers for finding appropriate resources based on the user's context information.

VI. Robustness Issues and Exception Handling Requirements

In our initial experiments with the implementation of the distributed meeting application we observed that the *Participant* role member who was performing classified presentation was able to continue it even after *Chairperson* role member left the room. This is a potential security violation. This situation arises because the precondition of the *StartClassifiedPresentation* operation is evaluated only once *before* the operation is initiated. For long running operation sessions this is inadequate as the context condition may become invalid while the session is still in progress. We consider such a security violation as an erroneous application behavior caused due to the change in environmental context that is required to remain valid for the duration of the execution of an operation session.

A. Exception Handling Requirements

We call the application errors due to changes in context conditions as *context invalidation errors*. In this paper we present two distinct approaches for handling such errors. There are other types of errors that arise in pervasive computing applications [16] which we do not address in this paper.

A role operation encounters a *context invalidation error* when the context condition associated with that role operation gets violated while the operation session is active. Such an error, occurring in *StartClassifiedPresentation* operation, can be handled in two ways. One approach is to automatically initiate an *alternate action* on some resource as part of error handling. Another approach is to perform *cooperative error handling* involving different members from one or more roles. For example, we could allow continuation of the presentation through some other operation after receiving appropriate approval from the secretary. Both these approaches are supported in our programming framework through the exception handling model as discussed below.

B. Exception Handling Model

We have proposed an exception handling model for pervasive computing applications in [16]. Here we present its implementation and demonstrate its utility for handling *context invalidation errors*. In this model two mechanisms are defined for exception handling: (i) exception handlers that are statically associated with role operations; and (ii) the set of exception operations, defined in the *exception interface* of a role. Exception handlers attached to role operations are used to specify automatic exception handling actions, while the *exception operations* in a role's *exception interface* are used for specifying exception handling actions that require participation of members from one or more roles for performing error recovery.

An exception handler for a particular exception type can be statically associated with a role operation, based on the *termination model*. Information about the exception occurrence, such as the role name, role member name, role operation name, and time is saved with each exception object. The exception handler contains an *action*, similar in syntax to the action specified for the role operation, which may specify execution of alternate actions on a resource or may explicitly signal the same or a different exception to the *exception interface* of the same or a different role.

Operations in the role's *exception interface* are similar to the normal role operations with one difference. An operation in the exception interface is only enabled when an exception of a particular type is delivered to the exception interface. This association between an exception and the operation is statically specified by the *WHEN* tag in the role's *exception interface*. The exception interface supports a queuing model for exception delivery and handling whereby the exception operation is enabled only when an exception is delivered to the *exception interface* queue and the exception is consumed when the associated operation is invoked.

C. *Exception Handling for Context Invalidation Errors*

The role operation specification is modified to include the *Context-Guard* tag through which the context-condition to be monitored is specified. A context guard specification consists of two things: specification of the trigger event, specified through the *WHEN* tag, and the specification of the condition to be monitored, specified through the *VALIDATION_CONDITION* tag. The context condition is evaluated whenever the trigger event is received. When the *VALIDATION_CONDITION* gets violated, it results in raising of the *ContextInvalidationException*, which causes the operation session to terminate. Below we present two approaches for handling *ContextInvalidationException* related to the *Participant* role's *StartClassifiedPresentation* operation.

1) Approach 1 - Alternative Action based Exception Handling: In Figure 11 we present the XML specification of the *StartClassifiedPresentation* operation that includes context-guard. The precondition and the action parts of the operation are the same as the corresponding parts in Figure 7. The context-guard is triggered by the occurrence of *RoomStatusChangeEvent* corresponding to the room in which the *Participant* is present and the classified presentation is in progress (line 5). The context condition to be monitored is specified in lines 6-9. The condition evaluates the presence of the *Chairperson* role member in *MyRoom* by invoking *isPresent* method on the corresponding room agent. The identity of the *Chairperson* role member is passed as the parameter to this method (line 7). The *ContextInvalidationException* is thrown by the runtime system if the validation condition is false, and the operation session is terminated. An exception handler for this exception is attached to the role operation (lines 11-16) which displays an unclassified presentation on *MyProjector* object (lines 12-15).

```
1.    <OPERATION NAME="StartClassifiedPresentation">
2.        <PRECONDITION> ... </PRECONDITION>
3.        <ACTION> ...</ACTION>
4.        <CONTEXT_GUARD>
5.            <WHEN EVENT_NAME="RoomStatusChangeEvent" FROM="MyRoom"/>
6.            <VALIDATION_CONDITION>
7.                <METHOD_INVOCATION OBJECT_REF="MyRoom" METHOD_NAME="isPresent"
                      ARG="members(Chairperson)">
8.                </METHOD_INVOCATION>
9.            </VALIDATION_CONDITION>
10.       </CONTEXT_GUARD>
11.       <ON_EXCEPTION NAME="ContextInvalidationException">
12.           <ACTION>
13.               <METHOD_INVOCATION OBJECT_REF="MyProjector" METHOD_NAME="display"
                      ARG="//Unclassified-Presentation">
14.               </METHOD_INVOCATION>
15.           </ACTION>
16.       </ON_EXCEPTION>
17.   </OPERATION>
```

Fig. 11. Specification of context-guard and *ContextInvalidationException* handler

2) Approach 2 - Cooperative Exception Handling: In this approach we require that the participant may be allowed to resume presentation, after the chairperson has left the room, only if *Secretary* role member approves of such a resumption and the chairperson has previously delegated such an approval to the secretary. Correspondingly, an operation to delegate the meeting control is required in the *Chairperson* role, and an operation to grant permission for resuming the presentation is required in the *Secretary* role. Furthermore we require that secretary's privilege to grant permission for resumption should be enabled only when *ContextInvalidationException* occurs in the *StartClassifiedPresentation* operation of the *Participant* role. To support this we need to propagate the *ContextInvalidationException* from the *Participant* role to the *Secretary* role and we need to define an operation in the *Secretary* role's exception interface that is enabled only when *ContextInvalidationException* is delivered to its exception interface queue. The exception handler that propagates the exception is shown below where the *TARGET* attribute specifies the target to which the exception should be sent:

```
<ON_EXCEPTION NAME="ContextInvalidationException">
   <ACTION> <SIGNAL TARGET="Secretary"/> </ACTION>
</ON_EXCEPTION>
```

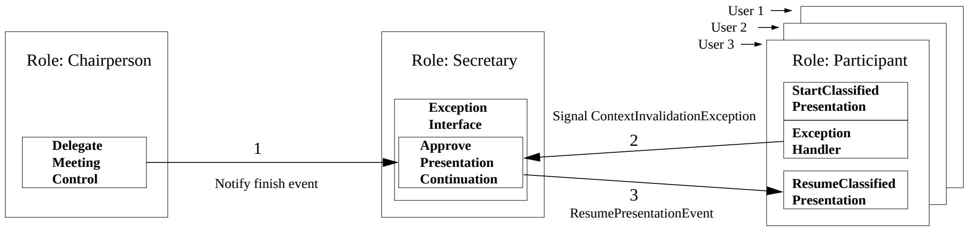

Fig. 12. Cooperative exception handling involving members from different roles

In Figure 12 we present a schematic of cooperative exception handling. The *Chairperson* role is provided with *DelegateMeetingControl* operation. By executing this operation the *Chairperson* role member delegates the meeting control to the *Secretary* role member (step 1). When the *ContextInvalidationException* occurs in the *StartClassifiedPresentation* operation invocation by any *Participant* role member, the attached exception handler explicitly signals it to the *Secretary* role (step 2). The *Secretary* role member approves the resumption, which results in notifying *ResumePresentationEvent* to the particular *Participant* role member (step 3). In Figure 13 we present the XML specification of the *Secretary* role. The *Secretary* role is provided with *ApprovePresentationContinuation* operation in its exception interface (lines 2-19). The *WHEN* tag associated with this exception operation (line 3) specifies that this operation is enabled when a *ContextInvalidationException* object is delivered to the exception interface queue. The precondition of this operation (lines 5-9) checks whether the count of *DelegateMeetingControl.finish* event is greater than zero indicating that the *Chairperson* has performed the *DelegateMeetingControl* operation. The action part of the operation (lines 10-16) consist of notifying the *ResumePresentationEvent*, specified through the *NOTIFY_EVENT* tag, to the particular participant whose *StartClassifiedPresentation* session had resulted in the *ContextInvalidationException*. The *grantee* attribute value inside this event is set by assigning the value obtained by invoking *getRoleMemberName* method on the *ContextInvalidationException* object, thus granting resumption permission to the particular participant who encountered the exception. The Figure 14 presents the specification of the *Participant* role's *ResumeClassifiedPresentation* operation. The precondition (lines 3-9) checks if the count of *ResumePresentationEvent.finish* event, in which the *grantee* attribute is set to *thisUser*, is greater than zero. This ensures that only the participant for whom the secretary granted the resumption permission is able to invoke this operation.

```
<ROLE NAME="Secretary">
    <EXCEPTION_INTERFACE>
        <WHEN EVENT_NAME="ContextInvalidationException">
            <OPERATION NAME="ApprovePresentationContinuation">
                <PRECONDITION>
                    <EVENT_CONDITION RELATION="GreaterThan" VALUE="0">
                        <EVENT_COUNT EVENT_TYPE="DelegateMeetingControl.finish" FROM="Chairperson">
                    </EVENT_CONDITION>
                </PRECONDITION>
                <ACTION>
                    <NOTIFY_EVENT NAME="ResumePresentationEvent" TARGET="Participant">
                        <SET_PARAM NAME="grantee">
                            <METHOD_INVOCATION OBJECT_REF="ContextInvalidationException"
                                METHOD_NAME="getRoleMemberName"/>
                        </SET_PARAM>
                    </NOTIFY_EVENT>
                </ACTION>
            </OPERATION>
        </WHEN>
    </EXCEPTION_INTERFACE>
</ROLE>
```

Fig. 13. *Secretary* role: *Exception interface* specification

```
<ROLE NAME=Participant>
    <OPERATION NAME="ResumeClassifiedPresentation">
        <PRECONDITION>
            <EVENT_CONDITION RELATION="GreaterThan" VALUE="0">
                <EVENT_COUNT EVENT_NAME="ResumePresentationEvent.finish" FROM="Secretary">
                    <EVENT_ATTRIBUTE NAME="grantee" RELATION="Equal" VALUE="thisUser"/>
                </EVENT_COUNT>
            </EVENT_CONDITION>
        </PRECONDITION>
        <ACTION>
            <METHOD_INVOCATION OBJECT_REF="MyProjector" METHOD_NAME="display"
                ARG="//Unclassified-Presentation"/>
        </ACTION>
    </OPERATION>
</ROLE>
```

Fig. 14. *Participant* role: *ResumeClassifiedPresentation* operation specification

VII. Related Work

Several other projects have developed middlewares for pervasive computing applications [1], [2], [3], [4], [5]. In [1] a programming model for building applications in active spaces is presented that enables selection of optimal resources for the task in the given environment. The main focus of the event translation middleware presented in [2] is to enable interoperability among various devices in pervasive computing environments. An adaptive middleware for pervasive computing applications, based on the negotiation of contracts between components, is presented in [3]. The Aura system [4] supports capturing user intent through representation of user tasks in pervasive computing applications. In [5] a middleware supporting context-sensitive interfaces, enabling context-aware application adaptation and ad-hoc communication, is presented.

The distinguishing aspect of our approach is automatic generation of a pervasive computing application environment through the integration of high-level application specification with a middleware. Our approach differs from all of the other projects along two dimensions: First, our programming framework supports multi-user coordination and security requirements for building context-aware pervasive computing applications involving multiple users. Second, we handle exceptions arising in pervasive computing applications through an exception handling model that we have developed and integrated in our programming framework. Among other systems, only [3] considers failures resulting due to change of application's components. However they do not consider context-sensitive security requirements and their violations arising due to changes in context conditions.

Other researchers have also looked at failures arising in pervasive computing environments [22], [23]. In [22] heart-beat based status monitoring, redundant provisioning of alternate services and applications, and restarting failed applications are proposed as possible failure handling approaches in pervasive computing applications. In contrast to this, we present a working application-level exception handling model integrated with our programming framework. In [23] a data-flow oriented architecture for building dependable pervasive computing systems is presented. The main objective of their design is to perform failure handling in pervasive computing applications *without* any user involvement. In contrast to this, human involvement is an integral part of our exception handling model.

There has been considerable work done on exception handling in workflow systems and office information systems [24], [25], [26], [27], [28], [29]. The traditional workflow and office information systems are fairly static. In contrast to this, context-aware pervasive computing applications are dynamic where resources are discovered and securely accessed based on the context information. This gives rise to the errors due to context failures which do not occur in the traditional workflow and office information systems. We handle such errors through our exception handling model.

Our programming framework supports dynamic security policies based on user's context in role-based collaborative applications. This role-based security model is more flexible than the standard NIST RBAC [30] model as it provides context-based permission activation and belongs to the family of active security models [31]. Other researchers have also considered context information as part of making access control decisions [32], [33], [34]. Our work differs from these in that we handle security violations due to dynamically changing context conditions through our context invalidation exception handler.

VIII. Conclusions

In this paper we have presented our programming framework for building pervasive computing applications that supports context-based resource discovery and binding, context-based proactive actions, and context-sensitive secure resource access. We have used this programming framework for building several pervasive computing applications such as *a distributed meeting*, *a shared whiteboard*, and *a context-aware examination environment*. In this paper we used the distributed meeting application to demonstrate the utility of this framework. We also presented integration of an agent-based distributed event monitoring system for collection and dissemination of context information. Lastly we identified errors occurring in the distributed meeting due to invalidation of context conditions and demonstrated two approaches for handling these errors using the exception handling model that we have integrated in our programming framework.

References

[1] A. Ranganathan, S. Chetan, J. Al-Muhtadi, R. Campbell, and M. Mickunas, "Olympus: A High-level Programming Model for Pervasive Computing Environments," in *Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications, PerCom 2005*, March 8-12 2005.

[2] R. Ballagas, A. Szybalski, and A. Fox, “Patch Panel: Enabling Control-Flow Interoperability in Ubicomp Environments,” in *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications*, March 14-17 2004, pp. 241–248.

[3] C. Becker, M. Handte, G. Schiele, and K. Rothermel, “PCOM - A Component System for Pervasive Computing,” in *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications, PerCom*, March 14-17 2004, pp. 67–76.

[4] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste, “Project Aura: Toward Distraction-Free Pervasive Computing,” *IEEE Pervasive computing*, vol. 1, no. 2, pp. 22–31, April-June 2002.

[5] S. S. Yau, F. Karim, Y. Wang, B. Wang, and S. K. S. Gupta, “Reconfigurable Context-Sensitive Middleware for Pervasive Computing,” *IEEE Pervasive Computing*, vol. 1, no. 3, pp. 33–40, 2002.

[6] N. Davies, K. Cheverst, K. Mitchell, and A. Efrat, “Using and Determining Location in a Context-Sensitive Tour Guide,” *IEEE Computer*, vol. 34, no. 8, pp. 35–41, 2001.

[7] M. Fleck, M. Frid, T. Kindberg, E. O’Brien-Strain, R. Rajani, and M. Spasojevic, “From Informing to Remembering: Ubiquitous Systems in Interactive Museums,” *IEEE Pervasive Computing*, vol. 1, no. 2, pp. 13–21, 2002.

[8] B. Schilit, N. Adams, and R. Want, “Context-Aware Computing Applications,” in *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, US, 1994, pp. 85–90.

[9] A. K. Dey, “Understanding and Using Context,” *Journal Of Personal And Ubiquitous Computing*, vol. 5, no. 1, pp. 4–7, 2001.

[10] K. Henricksen, J. Indulska, and A. Rakotonirainy, “Modeling context information in pervasive computing systems,” in *Pervasive ’02: Proceedings of the First International Conference on Pervasive Computing*. London, UK: Springer-Verlag, 2002, pp. 167–180.

[11] X. H. Wang, D. Q. Zhang, T. Gu, and H. K. Pung, “Ontology Based Context Modeling and Reasoning using OWL,” in *PERCOMW ’04: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*. Washington, DC, USA: IEEE Computer Society, 2004, p. 18.

[12] A. Tripathi, T. Ahmed, R. Kumar, and S. Jaman, “Design of a Policy-Driven Middleware for Secure Distributed Collaboration,” in *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS)*, July 2002, pp. 393 – 400.

[13] A. Tripathi, T. Ahmed, and R. Kumar, “Specification of Secure Distributed Collaboration Systems,” in *IEEE International Symposium on Autonomous Distributed Systems (ISADS)*, April 2003, pp. 149–156.

[14] A. Tripathi, T. Ahmed, D. Kulkarni, R. Kumar, and K. Kashiramka, “Context-Based Secure Resource Access in Pervasive Computing Environments,” in *1st IEEE International Workshop on Pervasive Computing and Communications Security (PerSec’04)*, March 2004, pp. 159–163.

[15] A. Tripathi, D. Kulkarni, and T. Ahmed, “A Specification Model for Context-Based Collaborative Applications,” *Elsevier Journal on Pervasive and Mobile Computing*, vol. 1, no. 1, pp. 21 – 42, May-June 2005.

[16] ——, “Exception Handling Issues in Context Aware Collaboration Systems for Pervasive Computing,” in *ECOOP Workshop on Exception Handling, Technical Report No 05-050, Department of Computer Science, LIRMM, Montpellier-II University*, vol. 4119. Springer LNCS, 2005.

[17] A. Tripathi, T. Ahmed, S. Pathak, M. Carney, and P. Dokas, “Paradigms for Mobile Agent-Based Active Monitoring,” in *Networks Operations and Management Symposium*, April 2002, pp. 65–78.

[18] A. R. Tripathi, M. Koka, S. Karanth, A. Pathak, and T. Ahmed, “ Secure Multi-agent Coordination in a Network Monitoring System,” in *Software Engineering for Large-Scale Multi-Agent Systems: Research Issues and Practical Applications, Springer, LNCS 2603* , January 2003, pp. 251 – 266.

[19] K. Kashiramka, “Semantic-Based Secure Resource Registration, Resource Discovery and Binding in Ubiquitous Computing Environments,” University of Minnesota, Twin Cities, Tech. Rep., December 2003, MS Plan B Report.

[20] T. Ahmed, “Policy Based Design of Secure Distributed Collaboration Systems,” Ph.D. dissertation, University of Minnesota, Twin Cities, 2004.

[21] 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*, vol. 62, pp. 123–140, 2002.

[22] S. Chetan, A. Ranganathan, and R. Campbell, “Towards Fault Tolerant Pervasive Computing,” *IEEE Technology and Society*, vol. 24, no. 1, pp. 38 – 44, 2005.

[23] C. Fetzer and K. Hogstedt, “Self*: A Data-Flow Oriented Component Framework for Pervasive Dependability,” in *Eighth IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 2003)*, Jan 2003.

[24] C. Hagen and G. Alonso, “Exception Handling in Workflow Management Systems,” *IEEE Transactions on Software Engineering*, vol. 26, no. 10, pp. 943–958, October 2000.

[25] F. Casati, S. Ceri, S. Paraboschi, and G. Pozzi, “Specification and Implementation of Exceptions in Workflow Management Systems,” *ACM Trans. Database Syst.*, vol. 24, no. 3, pp. 405–451, 1999.

[26] T. Murata and A. Borgida, “Handling of Irregularities in Human Centered Systems: A Unified Framework for Data and Processes,” *IEEE Transactions on Software Engineering*, vol. 26, no. 10, pp. 959–977, October 2000.

[27] D. K. W. Chiu, Q. Li, and K. Karlapalem, “Web Interface-driven Cooperative Exception Handling in Adome Workflow Management System,” *Inf. Syst.*, vol. 26, no. 2, pp. 93–120, 2001.

[28] D. M. Strong and S. M. Miller, “Exceptions and Exception Handling in Computerized Information Processes,” *ACM Trans. Inf. Syst.*, vol. 13, no. 2, pp. 206–233, 1995.

[29] H. Saastamoinen, “Survey on Exceptions in Office Information Systems,” in *Technical Report CU-CS-712-94 Department of Computer Science - University of Colorado, Boulder*, 1994.

[30] R. Sandhu, D. Ferraiolo, and R. Kuhn, “The NIST Model for Role-based Access Control: Towards a Unified Standard,” in *RBAC ’00: Proceedings of the fifth ACM workshop on Role-based access control*. New York, NY, USA: ACM Press, 2000, pp. 47–63.

[31] R. K. Thomas, “Team-based Access Control (TMAC): A Primitive for Applying Role-based Access Controls in Collaborative Environments,” in *RBAC ’97: Proceedings of the second ACM workshop on Role-based access control*. New York, NY, USA: ACM Press, 1997, pp. 13–19.

[32] C. K. Georgiadis, I. Mavridis, G. Pangalos, and R. K. Thomas, “Flexible Team-Based Access Control Using Contexts,” in *SACMAT ’01: Proceedings of the sixth ACM symposium on Access control models and technologies*. New York, NY, USA: ACM Press, 2001, pp. 21–27.

[33] G. Neumann and M. Strembeck, “An Approach to Engineer and Enforce Context Constraints in an RBAC Environment,” in *SACMAT ’03: Proceedings of the eighth ACM symposium on Access control models and technologies*. New York, NY, USA: ACM Press, 2003, pp. 65–79.

[34] M. J. Covington, W. Long, S. Srinivasan, A. K. Dey, M. Ahamad, and G. D. Abowd, “Securing Context-Aware Applications Using Environment Roles,” in *SACMAT ’01: Proceedings of the sixth ACM symposium on Access control models and technologies*. New York, NY, USA: ACM Press, 2001, pp. 10–20.