Building Context-Aware Healthcare Applications using a Generative Programming Framework

Devdatta Kulkarni and Anand Tripathi
Department of Computer Science
University of Minnesota Twin Cities
MN 55455, USA
Email: (dkulk,tripathi)@cs.umn.edu

***Abstract*—Developing context-aware applications is a tedious task which requires interfacing with different kinds of environmental sensors, new programming models, and extensive middleware support in the form resource discovery services, context management services, and context-based authorization services. We have developed a programming framework for building context-aware applications from their high-level design speci£cations. In this paper we show how this programming framework can be effectively used to build context-aware applications in the medical domain.**

I. Introduction

Recent trends in a number of application domains such as assisted living [1], [2], [3], hospital information systems [4], tour guides [5], [6], smart environments [7], is towards integrating context information to dynamically adapt an application's behaviour to provide enhanced functionality to users. Typical examples of context information include user location, co-location of users, co-location of a user with a device or an object, devices being used by a user, and so on. Context-aware applications, which may be single-user or collaborative multi-user, are immersed in sensor-rich environments. Context services collect sensor data from different kinds of environmental sensors and aggregate it to build high-level state information about the environment. Applications are designed to interface with such context services and dynamically adapt their behaviour, such as discovering and integrating application components and infrastructure services, based on the context information that is obtained through queries or noti£cations from context services.

We have developed a programming framework to support rapid construction of context-aware applications from their high-level speci£cations [8], [9]. The middleware generates the runtime environment of the application from the application's speci£cation. Middleware provides services for resource registration and discovery, location-independent naming, and context agents for detecting and aggregating sensor data. The main advantage of this approach is that the task of developing context-aware applications is simpli£ed because of the following reasons. First, the application programming efforts are limited only to developing the design speci£cation and the required application components. Modifying an application's design based on new requirements, stemming from user experience or technology evolution, is easy because it only involves changing the application's high-level design speci£cation. Second, the runtime environment for the application is automatically generated and maintained by the middleware, based on the design speci£cation.

In this paper we demonstrate how to use this programming framework for building medical domain applications. For this purpose we have emulated a patient information access application in our testbed laboratory environment. Context information could be used for different purposes in hospital information systems [10], [11], [4] and assisted living applications [1], [3]. These include, controlling access to information, recon£guring an application, notifying contextual alerts, coordinating user activities, and displaying information only relevant to a task in which a user is engaged. We elaborate on these requirements below.

- *Context-based access control:* This refers to using external world context information or application's internal context information for controlling user's access to resources/services. For example, we may require that a doctor may access patient records only from secure hospital locations [11].
- *Context-based recon£guration:* This refers to binding an application to resources/services based on

This work was supported by NSF grant 0411961.

context information. For example, health-care application on the nurse's mobile device may automatically bind to the records associated with the patient to whom she is currently attending.

- *Context-triggered actions:* This refers to automatically executing application tasks based on context information. For example, a hospital information system may support noti£cation of alerts to nurses when they enter a ward.
- *Context-based multi-user coordination:* This refers to utilizing context information as part of handling coordination and collaboration requirements in an application. For example, under special circumstances a nurse may request access to doctor's private notes about a patient from the patient's doctor. The nurse is able to access the records only after the doctor grants such an access.
- *Context-based information display:* This refers to modulating the information that is displayed to a user based on context. For elderly people, memory-loss is a signi£cant issue affecting their daily activities. Assisted living applications may help in such situations by providing context-based information display corresponding to the task that the person is engaged in [3]. For example, if an elderly person is engaged in the cooking activity then the application may display the actions performed by the person previously as part of this activity [3].

The paper is organized as follows. In Section II we present a brief overview of our programming framework. In Section III we present the design of the patient information access application. In Section IV we compare our programming model with other models for context-aware application development and conclude in Section V.

II. Conceptual Elements of the Programming Model

The environments in which a context-aware application is immersed are called "active space" [12]. Such spaces contain different kinds of sensors, infrastructure resources, services, and agents. A context-aware application may span across multiple such active spaces in different domains.

To model the design of a context-aware application deployed in such active spaces we have de£ned an abstraction called *activity* in the programming framework. An activity de£nes a shared object space, and a set of user *roles*. An activity may be distributed across different active spaces and it may involve multiple users in some collaborative tasks. Various resources/services required by the application are accessed as *objects* within the activity.

Roles: An activity contains zero or more roles. Each role contains one or more operations. An operation provides privileges for a role member to access objects de£ned within an activity. An operation contains an optional precondition and one or more actions. An action may specify one of the four things: (a) invoking a method on an object; (b) initiating an *interactive session* with an object involving any arbitrary sequence of a designated set of its methods; (c) binding an object to a resource/service; (d) signaling an application de£ned event.

Each event type de£nes an event stream. Filter operations can be de£ned on an event stream to derive its sub-stream satisfying certain predicates. The operator # on an event type returns the count of the events in its stream.

Context-based access control requirements are programmed using two mechanisms in our programming framework. These correspond to the *operation precondition* and the *access constraint*. The purpose of a precondition is to decide whether or not a role member should get access to a resource through a role operation. On the other hand, the purpose of an access constraint is to decide whether a role member should get access to a speci£c component of a resource. The information about such a component might be determined only during the operation's execution session and may not be known at the time of precondition evaluation.

Objects: An object serves as a reference within the activity for accessing a service or a resource. We have developed an XML schema called Resource Description De£nition (RDD) to describe objects. An RDD contains speci£cation of attribute-value pairs, interfaces, and events exported by the resource/service. An RDD is used as part of discovering a resource/service in an active space. Certain attributes in an object's RDD may be treated as *parameterized attributes*, meaning that their values are speci£ed at runtime, which may be based on the context information. These attributes are useful for performing context-based discovery and binding of objects with different resources and services.

Binding mechanisms are provided in the programming framework that specify *when* to perform a binding, *how* to identify the resource/service to be used in binding, and *what* context information, if any, should be used in the discovery process. There are two ways to identify a resource to be used in binding. One is to *directly* specify the resource/service URL, if it is known. Another is to *discover* the appropriate resource/service in the environment. Binding actions that are triggered by context events are performed whenever that event is noti£ed to the corresponding object manager by context agents.

Reactions: A reaction is used to perform event-

triggered actions in our programming model. Reactions may be used at the activity scope or at the scope of an object. Activity-level reactions are used for performing event-triggered actions that are relevant for the entire application. These include, notifying messages to different roles' members, and interconnecting different objects to connect their respective services to one another. Object-level reactions are used to program actions that are speci£c to an object, such as binding the object to an appropriate resource/service. There is an implementation level difference between the activity-level reactions and the object-level reactions. Each activity-level reaction is handled as a thread, whereas object-level reactions are maintained as binding policies with an object manager.

Middleware: The middleware provides three generic components, an *activity manager*, a *role manager*, and an *object manager*. The runtime environment of an activity is constructed by *deriving* policies from its XML speci£cation, and integrating them with the generic managers to construct application speci£c managers [13]. The policies that are derived include object binding and method level access control policies for object managers, operation execution and event subscription/noti£cation policies for role managers, and context event subscription policies for the activity manager. These managers are run on a set of *trusted servers*.

An object manager maintains a reference to the service to which it is currently bound. It enforces binding policies and method level access control policies. A role manager implements interfaces through which role management tasks such as, *joining* a role, *leaving* a role, or *selecting* a particular role member may be executed.

Users are admitted to various roles in an activity subject to the *role admission constraints*. Users execute role operations through an interface component called *User Coordination Interface (UCI)*. A UCI is generated for each user and it contains the GUI components for executing role operations corresponding to the roles in which the user is admitted. The UCI is transported to each user's device. Role operation executions are handled by the corresponding role manager.

III. Case Study: Programming Patient Information Access Application

Here we consider patient information access application in a hospital setting. It supports storage, retrieval, and access of patient information. Doctors and nurses may access this information through their mobile personal devices. We consider the following context-based requirements for this application.

- *Context-based Access control:* Doctors may access patient information only from secure hospital areas (requirement R1). Only those nurses and doctors who are on a patient's medical assistance team may access the patient's medical records [11], [14] (requirement R2).
- *Context-based Recon£guration:* The health-care application on the nurse's mobile device should automatically bind to the patient information resource corresponding to the patient to whom the nurse is currently attending (requirement R3).
- *Context-triggered Task Execution:* Nurses should be able to leave alerts for other nurses in each hospital ward. Such alerts should be noti£ed to a nurse when she enters a ward (requirement R4).
- *Context-based Multi-user Coordination:* A nurse should be able to request access to doctor's private notes about a patient from the patient's doctor. The nurse is able to access the records only after the doctor grants access to the nurse [11] (requirement R5).

We have programmed this application in our testbed environment. The emulated hospital environment consists of the following.

Infrastructure services and resources: We maintain a *patient database* to keep patient information. It contains three tables, *PatientTable*, *NurseTable*, *DoctorTable*. The *PatientTable* contains the £elds *patientID*, *teamID*, *DoctorNotes*, *NurseNotes*. The *NurseTable* contains the £elds *nurseID*, *teamID*. The *DoctorTable* has similar £elds. We also maintain a *message server* in the domain to store any messages/alerts.

Context agents: Context information is collected by context agents deployed in the environment, which are built using an agent-based distributed event monitoring and aggregation framework [15]. A *ward room agent* is associated with every room. It monitors user arrivals and departures from the room and generates user arrival events and user departure events. Such a monitoring is performed by tracking users' Bluetooth devices. There is also a *location agent* that maintains the current location of every nurse and doctor. One can also con£gure list of *secure locations* with this agent. In our testbed environment RFID tags are associated with every patient bed. The mobile devices given to nurses and doctors contains a RFID reader to sense the RFID tags.

Activity Speci£cation: For this application we de£ne the *PatientInformationAccess* activity. We show its partial speci£cation in the pseudo-notation below.

Activity PatientInformationAccess
  Role Doctor ..
  Role Nurse ..
  Object LocationAgent
    Action Bind LocationAgent **Direct** (//LocationAgentURL)
  Object PatientDBService
    Action Bind PatientDBService **Direct** (//PatientBDServiceURL)

```
Object MessageServer
    Action Bind MessageServer
        Direct (//MessageServerURL)
Object NurseUCIObj
```

We de£ne two roles, *Doctor*, and *Nurse* in this activity. Multiple users may be admitted to each role. We de£ne four objects, *LocationAgent*, *PatientDBService*, *MessageServer*, and *NurseUCIObj* in this activity. We bind the £rst three objects at the activity instantiation time to the corresponding services, whereas the *NurseUCIObj* is bound at runtime.

1. Context-based Access Control: Here we present speci£cation that satis£es requirements R1 and R2.

a) Precondition model: To satisfy requirement R1 we want to restrict a doctor's access to patient records to happen only from secure locations. For this, we need to check whether the current location from which the *Doctor* role member is accessing patient records is secure or not. This can be done by querying the *LocationAgent* as part of operation precondition as shown below.

```
Role Doctor
    Operation AccessPatientData
        Precondition LocationAgent.isSecureLocation(thisUser)
        Action PatientDBService
            SessionMethod accessPatientRecords
```

The operation *AccessPatientData* is provided for the *Doctor* role through which patient information may be accessed. The access control requirement is programmed as the precondition of this operation. We query the *LocationAgent* to check whether the doctor's current location belongs to the list of known secure locations. As part of this operation we want to allow only the *accessPatientRecords* method to be invoked on the *PatientDBService* object. Hence in the *SessionMethod* tag we specify only this method's name. In our programming model *thisUser* is a special variable which refers to the role member who is invoking the operation.

b) Access constraint model: To satisfy requirement R2 we want to restrict a nurse's access to only those patients' records on whose team she is present. For this, we need to compare the *teamIDs* of the teams on which the *Nurse* role member is present with the *teamID* of every patient. This checking can be done only *after* an operation's execution has begun. Precondition model is not suitable for this purpose because the resource component to be accessed may not be known at the precondition evaluation time. Instead, the access constraint is appropriate for this purpose. An access constraint is evaluated at runtime and is used to specify *content-based* access control requirements. We show this speci£cation below.

```
Role Nurse
    Operation AccessPatientRecords
        Action PatientDBService SessionMethod
            viewPatientRecords addNurseNotes
        AccessConstraint (teamID =
            PatientDBService.getNurseTeamID(thisUser))
```

The operation *AccessPatientRecords* is provided for the *Nurse* role to access patient records. The access control requirement is programmed as the *AccessConstraint* speci£cation for this operation. It ensures that only those patients' records whose *teamID* matches any of the *teamIDs* of the *Nurse* role member are accessible through the session methods.

2. Context-based resource binding: As part of requirement R3 we want that when a nurse is attending to a patient, the application on the *Nurse*'s mobile device should bind to that particular patient's records. This is useful to ensure that a nurse does not mistakenly access records of a wrong patient. Below we show this speci£cation.

```
Role Nurse
    Object PatientRecord RDD PatientInfo.xml
        Reaction When Event thisDevice.RFIDEvent
            Action Bind PatientRecord Discover
                (patientID=RFIDEvent.getID())
    Operation AccessRecord
        Action PatientRecord SessionMethod displayInformation
```

We de£ne the *PatientRecord* object for this purpose. We specify the *PatientInfo.xml* as the RDD speci£cation for this object. It contains attribute-value pairs related to a patient, such as *patientID*, *date-of-entry*, and so on. We specify that the binding of this object should be triggered by the *RFIDEvent*. This event is generated by the RFID detector service running on nurse's mobile device. We specify discovery based binding for this object. The *patientID* £eld in the RDD is £lled in at runtime by querying the *RFIDEvent*. This is a *parameterized attribute* in this RDD. The completed RDD is used to query the *PatientDBService* and the object is bound to the returned reference.

3. Context-triggered actions: As part of requirement R4 we want that the *Nurse* role member be able to post alerts speci£c to a ward. Other nurses are noti£ed of such alerts when they enter the ward. Below we show this speci£cation.

```
Role Nurse
    Operation PostWardAlerts
        Action MessageServer SessionMethod writeAlert
            AccessConstraint
                (location=LocationAgent.getLocation(thisUser))
Reaction NotifyAlerts
    When LocationAgent.NurseArrivalEvent
    Condition MessageServer.isMessageOutStanding(
        NurseArrivalEvent.getLocation())
    Action Bind NurseUCIObj With
        getUCIObject(NurseArrivalEvent.getUser())
    Action NurseUCIObj.notifyAlert(MessageServer.getMessages(
```

NurseArrivalEvent.getLocation()))

We de£ne the *PostWardAlerts* operation through which a nurse may write alerts by invoking the *writeAlert* method as part of the session on the *MessageServer*. Alerts are stored as records indexed by location on the message server. We use access constraint to select only those records for which the *location* attribute value matches the nurse's current location and only these records are updated.

We also de£ne the *NotifyAlerts* reaction in the activity. This reaction is triggered by the *NurseArrivalEvent* that is generated whenever any nurse enters any ward. As part of the reaction's condition evaluation we check whether any message is outstanding corresponding to the current location. If so, then we notify all such messages to the nurse's UCI. This is achieved by £rst binding the *NurseUCIObj* to the UCI of the *Nurse* role member corresponding to whom the event is generated and then notifying the messages to that UCI. The method *getUCIObject* is provided in the programming framework which provides the reference to a role member's UCI.

4. Context-based multi-user coordination: As part of requirement R5 we need the following. A *Nurse* role member should be able to request access to a particular patient's records from a speci£c doctor. The doctor from whom the access is requested should be able to grant access to the records of the requested patient and also to only the nurse who has requested the access.

We achieve this using application de£ned events to coordinate the *Doctor* and *Nurse* role members. We de£ne the *RequestAccessEvent* to indicate the *Nurse* role member's request to access a particular patient's records. The following attributes are set in this event, *NurseID*, *DoctorID*, and *PatientID* indicating the access request. This event is noti£ed from the *Nurse* role to the *Doctor* role. We also de£ne the *AccessApprovalEvent* to indicate the *Doctor* role member's approval given to a particular *Nurse* role member corresponding to a speci£c patient. This event is noti£ed from the *Doctor* role to the *Nurse* role. Below we show this speci£cation.

```
1.Role Nurse
2.  Operation RequestAccess
3.      Action NotityEvent RequestAccessEvent
4.          (NurseID=thisUser, DoctorID=Doctor.selectMember(),
5.          PatientID=PatientDBService.selectPatient())
6.  Operation AccessPatientData
7.      Precondition
8.          #AccessApprovalEvent(nurseID=thisUser) > 0
9.          Action PatientDBService
10.             SessionMethod displayPatientInformation
11.             AccessConstraint
12.             (patientID=AccessApprovalEvent(nurseID=thisUser).
13.             getAttribute(patientID))
14.Role Doctor
15. Operation ApprovePatientDataAccess
16.     Precondition
17.         #RequestAccessEvent(DoctorID=thisUser) > 0
18.         Action NotifyEvent AccessApprovalEvent
19.         (nurseID=RequestAccessEvent.getAttribute(NurseID),
20.         patientID=RequestAccessEvent.getAttribute(PatientID))
```

In the *Nurse* role we de£ne the *RequestAccess* operation (lines 2-5) through which a *Nurse* role member may request access. In the *Doctor* role we de£ne the *ApprovePatientDataAccess* operation (lines 15-20) through which a *Doctor* role member may grant approval to the requesting *Nurse* role member. A *Doctor* role member is able to execute this operation only if the count of *RequestAccessEvent* corresponding to that *Doctor* role member is non-zero. This is speci£ed through the operation's precondition. The construct *(DoctorID=thisUser)* is a £lter predicate that selects only those events that correspond to the *Doctor* role member who is invoking the operation. As part of this operation's action an instance of *AccessApprovalEvent* is signaled. The attributes *nurseID* and *patientID* in this event are set by querying the values of attributes *NurseID* and *PatientID* respectively from the *RequestAccessEvent*.

In the *Nurse* role we de£ne the *AccessPatientData* operation (lines 6-13) through which a *Nurse* role member may access patient information. The precondition of this operation ensures that only the *Nurse* role member to whom the *Doctor* role member granted permission is able to invoke this operation. Furthermore, the access constraint speci£cation ensures that the *PatientDBService* object manager grants access to only those patient records for whom the *Doctor* role member gave access through the *AccessApprovalEvent*.

IV. Related Work

Several other research groups have developed programming frameworks for context-aware applications [16], [17], [18], [19], each focusing on different levels of system abstractions. Gaia [16] seeks to provide operating system level support for active space applications. RCSM [17] concentrates on interface definition language level support, Chisel [18] focuses on reusable policies for context-aware applications, whereas PCOM [19] focuses on supporting component integration based on inter-component contracts. In comparison to other high-level programming models, our focus is on generating complete runtime environment of context-aware applications from their high-level speci£cations. The distinguishing aspect of our work is that the entire operational con£guration of a context-aware application, under different context conditions, is laid out in the form of its design speci£cation.

Access control requirements and models in health-care applications have been studied by others [14], [10], [11].

In [14] a *team* abstraction is used to perform access control decisions related to a patient's medical information. In the OASIS RBAC model [10], parameterized role membership certi£cates are used to enforce active security requirements for the Electronic Health Records (EHR) access system.

Our context-based access control mechanisms differ from these in the following two ways. First, the notion of events and their integration with authorization mechanisms is a salient feature of our RBAC model. Event-based predicates are used to explicitly specify dynamic constraints in our framework. This model has evolved from our experiences in building CSCW applications using it [20]. Second, through the *access constraint* mechanism we are able to enforce access control requirements using resource's attributes and role member's context for this purpose. Attribute-based access control falls under the requirement of *content-based access control*. Content-based access control ideas can be traced back to early work on access control based on *data types* in programming languages [21].

V. Conclusions

In the coming years there will be considerable interest in developing context-aware medical domain applications given the cheap availability of different kinds of sensors and also HIPAA's stipulation [22] of "physical safeguards" for health information systems as part of its security rules. Building context-aware applications requires new programming models which support integration of context information for driving context-based application adaptations in terms of resource usage, access control, automated task executions, and multi-user coordination. Our high-level programming framework for context-aware applications supports mechanisms for programming such adaptive capabilities in context-aware applications. In this paper we demonstrated that this framework can be used to design and implement medical domain applications having unique context-based characteristics.

References

[1] S. Consolvo, P. Roessler, B. E. Shelton, A. LaMarca, B. Schilit, and S. Bly, "Technology for care networks of elders," *IEEE Pervasive Computing*, vol. 3, no. 2, pp. 22–29, 2004.

[2] S. Helal, B. Winkler, C. Lee, Y. Kaddoura, L. Ran, C. Giraldo, S. Kuchibhotla, and W. Mann, "Enabling location-aware pervasive computing applications for the edlerly," in *IEEE PERCOM*. Washington, DC, USA: IEEE Computer Society, 2003, p. 531.

[3] E. D. Mynatt, A.-S. Melenhorst, A. D. Fisk, and W. A. Rogers, "Aware technologies for aging in place: Understanding user needs and attitudes," *IEEE Pervasive Computing*, vol. 03, no. 2, pp. 36–41, 2004.

[4] J. E. Bardram, T. R. Hansen, M. Mogensen, and M. Søgaard, "Experiences from real-world deployment of context-aware technologies in a hospital environment," in *Ubicomp*, 2006, pp. 369–386.

[5] 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, August 2001.

[6] G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton, "Cyberguide: a Mobile Context-aware Tour Guide," *Wirel. Netw.*, vol. 3, no. 5, pp. 421–433, 1997.

[7] Y. Shi, W. Xie, G. Xu, R. Shi, E. Chen, Y. Mao, and F. Liu, "The smart classroom: Merging technologies for seamless tele-education," *IEEE Pervasive Computing*, vol. 02, no. 2, pp. 47–55, 2003.

[8] A. Tripathi, D. Kulkarni, and T. Ahmed, "A Speci£cation Model for Context-Based Collaborative Applications," *Elsevier Journal on Pervasive and Mobile Computing*, vol. 1, no. 1, pp. 21 – 42, May-June 2005.

[9] D. Kulkarni and A. Tripathi, "Generative programming approach for building pervasive computing applications," in *SEPCASE '07: Proceedings of the 1st International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments*. Washington, DC, USA: IEEE Computer Society, 2007, p. 3.

[10] J. Bacon, K. Moody, and W. Yao, "A model of oasis role-based access control and its support for active security." *ACM Trans. Inf. Syst. Secur.*, vol. 5, no. 4, pp. 492–540, 2002.

[11] M. Evered and S. Bögeholz, "A case study in access control requirements for a health information system," in *ACSW Frontiers '04: Proceedings of the second workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation*. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2004, pp. 53–61.

[12] M. Román, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt, "Gaia: a middleware platform for active spaces." *Mobile Computing and Communications Review*, vol. 6, no. 4, pp. 65–67, 2002.

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

[14] R. K. Thomas, "Team-based Access Control (TMAC): A Primitive for Applying Role-based Access Controls in Collaborative Environments," in *ACM Workshop on Role-based Access Control*, 1997, pp. 13 – 19.

[15] A. R. Tripathi, D. Kulkarni, H. Talkad, M. Koka, S. Karanth, T. Ahmed, and I. Osipkov, "Autonomic Con£guration and Recovery in a Mobile Agent-based Distributed Event Monitoring System ," *Softw. Pract. Exper.*, vol. 37, no. 5, pp. 493–522, 2007.

[16] A. Ranganathan, S. Chetan, J. Al-Muhtadi, R. H. Campbell, and M. D. Mickunas, "Olympus: A High-Level Programming Model for Pervasive Computing Environments," in *PerCom*, 2005, pp. 7–16.

[17] S. S. Yau, F. Karim, Y. Wang, B. Wang, and S. K. S. Gupta, "Recon£gurable Context-Sensitive Middleware for Pervasive Computing," *IEEE Pervasive Computing*, vol. 1, no. 3, pp. 33–40, 2002.

[18] J. Keeney and V. Cahill, "Chisel: A Policy-Driven, Context-Aware, Dynamic Adaptation Framework," in *POLICY '03: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 3–14.

[19] C. Becker, M. Handte, G. Schiele, and K. Rothermel, "PCOM - A Component System for Pervasive Computing," in *IEEE PERCOM*, March 14-17 2004, pp. 67–76.

[20] T. Ahmed and A. R. Tripathi, "Speci£cation and veri£cation of security requirements in a programming model for decentralized cscw systems," *ACM Trans. Inf. Syst. Secur.*, vol. 10, no. 2, p. 7, 2007.

[21] A. K. Jones and B. H. Liskov, "A language extension for expressing constraints on data access," *Commun. ACM*, vol. 21, no. 5, pp. 358–367, 1978.

[22] "HIPAA Security Series," Available at URL http://www.cms.hhs.gov/EducationMaterials/.