Support for Situation-Awareness in Trustworthy Ubiquitous Computing Application Software

Stephen S. Yau, Dazhi Huang, Haishan Gong, Yisheng Yao

Department of Computer Science and Engineering

Arizona State University

Tempe, AZ 85287-8809, USA

{yau, dazhi.huang, haishan.gong, yisheng.yao}@asu.edu

Abstract

Due to the dynamic and ephemeral nature of ubiquitous computing (ubicomp) environments, it is especially important that the application software in ubicomp environments is trustworthy. In order to have trustworthy application software in ubicomp environments, situation-awareness (SAW) in the application software is needed for enforcing flexible security policies and detecting violations of security policies. In this paper, an approach is presented to providing development and runtime support for incorporating SAW in trustworthy ubicomp application software. The development support is to provide SAW requirement specification and automated code generation for achieving SAW in trustworthy ubicomp application software, and the runtime support is for context acquisition, situation analysis, and situation-aware communication. To realize our approach, the improved Reconfigurable Context-Sensitive Middleware (RCSM) is developed for providing the above development and runtime support.

Keywords: Trustworthy ubiquitous application software, situation-awareness, Situation-Aware Interface Definition Language (SA-IDL), Situation-Aware (SA) middleware, SA security policies, development and runtime support.

1. Introduction

The objective of ubiquitous computing (ubicomp) is to provide access to information and computing resources for users at any time and anywhere, and make the underlying computing and communication techniques transparent to users. Ubicomp environments can be considered as a collection of embedded, handheld, wearable, and mobile devices, all connected to each other through wireless or wired networks and acting collaboratively to perform certain tasks for users with little or no user attentions. Due to the dynamic and ephemeral nature of ubiquitous computing environments, it is especially important that the application software in ubicomp environments is trustworthy under various situations. Thus, application software running on the mobile and wearable devices in ubicomp environments often needs to be *Situation-Aware* (SA), i.e., the application software should have the capability of being aware of situations and adapting devices' behavior based on situation changes. A *situation* is a set of contexts over a period of time that is relevant to future device actions. A *context* is any instantaneous, detectable, and relevant property of the environment, the system, or users, such as time, location, light intensity, available bandwidth and a user's schedule [1]. Contexts may be sensed and extracted by a set of sensors deployed in the environment or connected to mobile devices. *Situation-awareness* (SAW) is essential for trustworthy ubicomp application software because it is needed for enforcing flexible security policies and detecting violations of security policies. However, incorporating SAW in

trustworthy ubicomp application software is challenging due to the severe resource constraints of ubicomp devices and the ephemeral and heterogeneous ubicomp environments. Hence, development and runtime support for SAW is needed to simplify the development and improve the performance of trustworthy ubicomp application software.

In this paper, we will identify the development and runtime support for SAW in trustworthy ubicomp application software, and present our approach to providing such support using SA middleware. An example will be given to illustrate our approach.

2. An Illustrative Example

Before presenting our approach, we would like to discuss the following example to show the need for SAW in trustworthy ubicomp application software and how SAW can be utilized to provide trustworthiness, especially in enforcing flexible security policies and detecting violations of security policies.

Consider a Smart Classroom, in which students and instructors use their PDAs (or tablet PCs) to engage in activities, such as gaining accesses to equipment in the classroom, group discussions, and course material dissemination [2]. Some PDAs are connected to sensor units which collect context data from ambient environments. The following scenario shows some operations of ubicomp application software in Smart Classroom:

i. A door with embedded processor and sensors is used to control the access to the Smart Classroom. Sensor data is collected every second. Each person will carry an RFID tag storing the person's identification and role (instructor/student). When a sensor on the door detects a person standing in front of the door for three seconds, the door controller checks whether the person has the permission to enter the classroom, and will open the door if permitted. The door controller can retrieve the information of the course which is going to be held or being held in the classroom from a course monitor, regarding the course time and the valid RFID list. The access control policy, "Instructors and students of a class are allowed to enter the classroom 15 minutes before their class starts or during the class" needs to be enforced by the door controller.

ii. Due to the fact that several persons may enter the classroom together, another sensor, classroom monitor, on the door is used to count the number of persons entering or leaving the classroom. The ubicomp application software in smart classroom also keeps track with different RFID signals in the classroom. When the number of persons in the classroom is greater than the number of RFID signals in the classroom, it means that at least one person without RFID tag is in the classroom, or at least one person in the classroom wearing an invalid RFID

tag. The door controller needs to identify such a violation of the access control policy in (i), and notifies the instructor that there are some persons in the classroom without proper authorization by triggering the alarm.

In this example, a person can enter classroom only under a specific situation of "allow entering classroom", defined by time, location, user's identification and user's action. When the situation is changed, for example passing the class time or wearing an invalid RFID, the person will not be able to enter the classroom. The alarm is triggered under the situation of "intruder in classroom", which means that an unauthorized person entered the classroom. This example shows how SAW can be used to enforce flexible access control policies (see i), from which access decisions are dynamically changing when situation changes, and to detect violations of the access control policies (see ii) in ubicomp application software.

3. Development and Runtime Support for SAW in Trustworthy Ubicomp Application Software

Trustworthy ubicomp application software usually operates with the three phases shown in Figure 1:

Phase 1. Acquiring from ambient environments the relevant contexts, which constitue the situations triggering proper actions of the software.

Phase 2. Recognizing situations that will trigger any actions of the software.

Phase 3. Triggering proper actions for enforcing SA security policies based on the current situation.

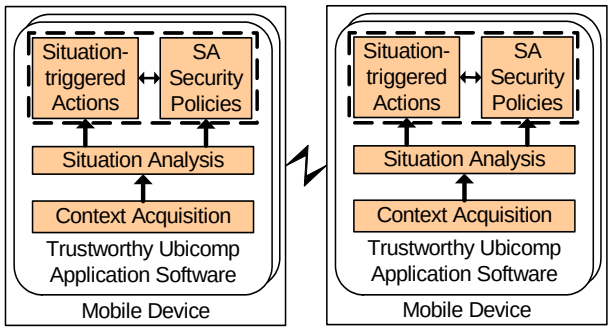

Figure 1. Execution process of trustworthy ubicomp application software

In Phase 3, situation-triggered actions often need to be performed in real-time, and require proper scheduling if multiple actions need to be triggered simultaneously. Furthermore, the situation-triggered actions in one trustworthy ubicomp application software may require spontaneously establishment of communication channels with other trustworthy ubicomp application software. This type of communication is called *SA communication* [3] since the establishment of communication channels is triggered under certain situations.

From these three phases, besides the capability of security policy enforcement, trustworthy ubicomp application software needs to have the capabilities of context acquisition, situation analysis, SA communication and action scheduling. Due to the severe resource constraints of ubicomp devices and the dynamic changing ubicomp environments, the development and runtime support should simplify the development and improve the performance of trustworthy ubicomp application software with these capabilities, and facilitate the runtime reconfiguration of the

software to adapt to the changing ubicomp environments. Hence, the following development support for SAW in trustworthy ubicomp application software is desirable:

D1. Generate specification of situations of interests and relevant contexts.

D2. Generate specification of actions to be triggered with their real-time requirements under certain situations.

D3. Verify SAW requirements of the software.

D4. Automated code generation of the components for SAW the software.

D5. Test automation for the SAW aspect of the software.

The following runtime support is required for SAW in trustworthy ubicomp application software:

R1. Transparent and on-demand context acquisition with well-defined interfaces for the software to retrieve contexts from various sources.

R2. Situation analysis to recognize the situations of interest based on situation specifications.

R3. Situation-based action triggering and scheduling to satisfy their real-time requirements.

R4. SA communication for spontaneous establishment of communication channels.

The above desirable development support should reduce the development effort of the software and greatly improve the reusability of the software. The above required runtime support should consume little resources, such as energy, memory, and CPU time of ubicomp devices, and allow runtime reconfiguration of SAW requirements.

4. Current State of the Art

Substantial research has been done in context-aware computing in ubicomp environments, including some frameworks, toolkits and infrastructures for providing support to situation- or context-aware application development. CALAIS [4] focuses on applications accessible from mobile devices and supports acquisition of context about users and devices, but it is difficult to evolve existing applications when requirements for context acquisition and the capabilities and availabilities of sensors change. Context Toolkit [5] provides architectural support for context-aware applications, but does not provide analysis of complex situations, which may need multiple contexts over a period of time. CoolTown [6] supports applications that display contexts and services to end-users. MobiPADS [7] is a reflective middleware designed to support dynamic adaptation of context-aware services based on which application's runtime reconfiguration is achieved. GAIA [8] provides context service, space repository, security service and other QoS for managing and interacting with active spaces. TSpaces [9] utilizes tuple spaces to store contexts and allows tuple space sharing for application software to read and write, but it ignores

the status of the devices where the application software executes, network conditions, and the surrounding environment as part of the overall context. However, none of these approaches support context discovery, SA communication and runtime modification of context- or situation-awareness requirements. We have developed a middleware, Reconfigurable Context-Sensitive Middleware (RCSM) [1, 20, 28], which provides some essential runtime and development support, including automated generation of Adaptive Object Containers (ADCs) for situation analysis, transparent context acquisition, SA communication and situation-based action triggering for SA ubicomp application software. However, it does not provide on-demand context acquisition and action scheduling, not allow runtime modifications of SAW requirements, and not support sharing situation information among multiple SA ubicomp application software. In this paper, we will present a new middleware design, which will overcome these difficulties of our RCSM [1, 20, 28].

Many access control mechanisms, such as mandatory access control (MAC) [10], discretionary access control (DAC) [11], and role-based access control (RBAC) [12], have been developed. However, the highly dynamic nature of trustworthy ubicomp application software imposes additional requirements on the security mechanisms, such as providing support of least privilege in dynamic environment, and reducing management overhead, which are not solvable by these classic access control mechanisms. In MAC, the security labels of subjects and objects are relatively static. MAC does not sufficiently address least privilege in dynamic systems and offers little support for message integrity. In DAC, whoever is authorized to access an object may do whatever with the object as he likes, including delegating the access rights to others, or using it for some purpose which may violate the system security requirements. The combination of DAC and MAC overcomes most of these difficulties, but the large management overhead makes it not suitable for trustworthy ubicomp application software. In RBAC, permissions are assigned to roles and roles are assigned to users. When the assignment of permissions to roles is relatively stable, adopting RBAC significantly reduces the management overhead. If the concept of a role is relatively unstable, as in trustworthy ubicomp application software, the assignment of permissions to roles is not as static as desired, and hence the advantage of using RBAC in its original form [12] is very limited.

Recently, a general temporal RBAC (GTRBAC) [13] has been developed to incorporating temporal constraints to activate and enable roles such that more flexible policies can be specified to address least privilege and management overhead. However, other non-temporal context constraints, such as location and network bandwidth, also need to be incorporated in trustworthy ubicomp application software for specifying and managing security policies easily.

So far, several context-based access control systems [14, 15] have been developed for ubicomp applications. Covington *et al* [14] defined environment contexts as environment roles, and permissions are assigned to the subjects if the environment roles are evaluated as true based on the current context value. However, it is not easy to use environment roles to express the status of subjects and objects, which is often needed in access control. For example, the current access right of a user may depend on whether the user has been successfully authenticated by someone in the system. Cholweka *et al* [15] developed a context-sensitive access control model, in which the access rights of a subject are granted based on the actual task of the subject. These context-based access control approaches [14, 15] can be considered as special cases of our approach of SA security policy specification and enforcement, which will be presented in Section 8 of this paper, since the definition of situation is broader than the context in [14, 15].

5. Modeling SAW in Trustworthy Ubicomp Application Software

To facilitate the analysis and specification of SAW requirements of trustworthy ubicomp application software, we have developed an ontology for modeling SAW in the software. Our SAW ontology describes the essential entities for representing SAW and the relationships among these entities. The advantages of our SAW ontology are that it describes an abstract and application-independent view of SAW, and can be easily shared or extended to model SAW in different application domains. Figure 2 shows the following entities in our SAW ontology:

- A *context* has a *context name* and a *context type*.
- A *context value domain* is defined for each *context type* and *context operators* are defined on *context value domains*.
- An *argument* could be a constant, a variable of specific context type, or a context value at a particular time.
- A *context operator* is an operator used for calculating or comparing arguments involved in an atomic situation.
- An *atomic situation* is a situation defined using

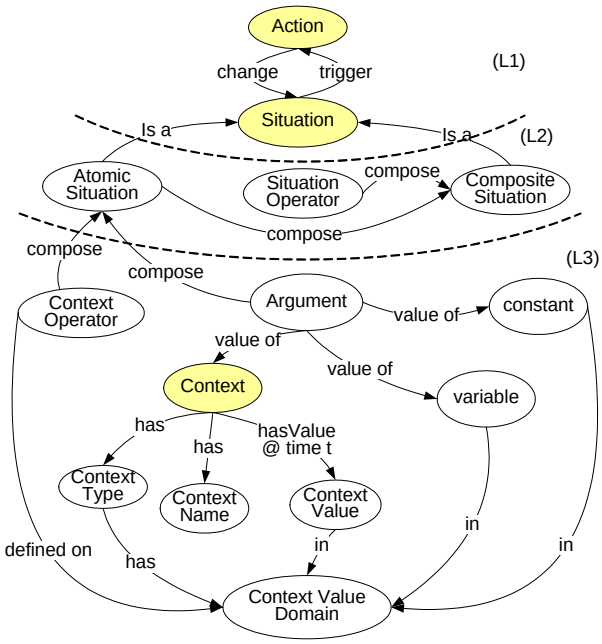

Figure 2. Our ontology for modeling SAW in trustworthy ubicomp application software

context operators and arguments, and cannot be decomposed into any other atomic situations.

- A *situation operator* is a set of logical and temporal operators.
- A *composite situation* is a situation recursively composed by atomic situations and other composite situations using situation operators.
- *Actions* will be triggered in a particular order determined by their priorities and execution deadlines, when a situation is recognized. The execution of actions normally causes a situation change.

Our SAW ontology is organized in three layers describing a top-down process of SAW requirement analysis by developers of trustworthy ubicomp application software. Layer L1 captures the triggering relations between situations and actions and corresponds to the first step of SAW requirement analysis, where a developer identifies a set of situations and actions of interest. Layer L2 shows how a composite situation is composed by a set of atomic situations and situation operators. In this step, developers need to decompose each of the identified situations into a set of atomic situations based on their temporal or logical relations. Layer L3 depicts the relevant contexts and the operations on these contexts for recognizing an atomic situation. In this step, developers analyze each atomic situation by identifying contexts, context related properties and context operators.

Our SAW ontology helps developers capture sufficient SAW information through requirements analysis. It also provides a basis from which to build a specification language with good expressiveness to facilitate developers to easily specify the SAW requirements. The relations among entities in our SAW ontology define the process of situation analysis for recognizing a situation. Given SAW specifications, the process of situation analysis can be automatically done. We will show in Section 7.1 that such a specification language provides desirable development support D1 and D2, and in Section 6.2 that our approach to situation analysis provides required runtime support R2. A tool for automated code generation for trustworthy ubicomp application software is developed to provide required runtime support D4.

6. Design of a Middleware for Incorporating SAW in Trustworthy Ubicomp Application Software

Middleware is an effective approach to providing the required runtime support due to the following reasons:

a) Middleware provides a set of low-level components effectively addressing the needs for SAW in trustworthy ubicomp application software. These components are shared and reused by various trustworthy ubicomp application software, and hence greatly reduce the cost of software development and improve the quality of the developed software.

b) Middleware masks the complexity of underlying hardware and software platforms by providing a set of well-defined interfaces to the low-level components for SAW, which allows developers to focus on high-level functionality of application software without worrying low-level details, such as contexts acquisition from low-level sensing units and situation analysis.

c) By separating the functional components of trustworthy ubicomp application software from its low-level components that provide runtime support for SAW, the developed software is easier to evolve since changes of the functional components and the low-level components will not affect each other.

d) Middleware can seamlessly integrate SAW and security of ubicomp application software to achieve trustworthiness. Existing techniques for middleware security [17, 18] have already shown how middleware can seamlessly integrate security capabilities into distributed applications based on existing security mechanisms. Given a suitable approach that integrates SAW and security aspects, such as our SA access control model [16], middleware will be an effective tool to integrate SAW and security to satisfy the need of trustworthy ubicomp application software for more dynamic security policy management and enforcement.

e) Middleware can greatly improve interoperability of various ubicomp devices.

In Subsections 6.1 and 6.2, we will present our design of a middleware providing runtime support R1-R4 for SAW based on our SA communication and context discovery protocols [3, 19-21]. This middleware is a distributed object middleware [22] since trustworthy ubicomp application software often consists of a set of application objects interacting with other objects (locally or remotely). We call this middleware as *Situation-Aware (SA) middleware*, and the application objects of the software with SAW requirements as *SA objects*.

6.1 Architecture of Our SA Middleware

Figure 3 shows the architecture of our SA middleware, which runs on each ubicomp device, where a trustworthy ubicomp application software is

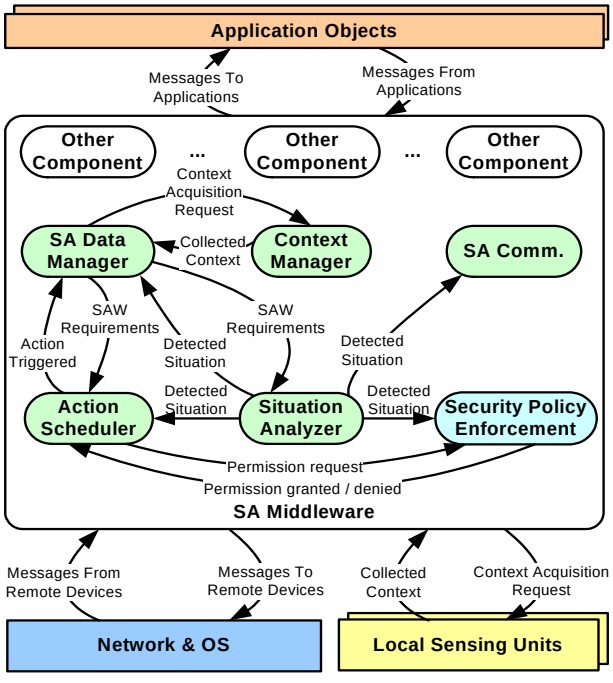

Figure 3. Architecture of our SA middleware

running. Multiple trustworthy ubicomp application software can run simultaneously on top of this SA middleware. This SA middleware collaborates with SA middleware running on other devices to discover remote application objects and context sources. Runtime support R1-R4 is provided by the following four components in our SA middleware:

- *Context Manager* providing R1. It receives context acquisition requests from Situation Data Manager, and sends collected context data to Situation Data Manager. The design of Context Manager is partly based on the context management module in our RCSM. An important new capability provided by Context Manager is context discovery, which enables on-demand context acquisition.
- *Situation Analyzer* providing R2. It recognizes situations based on the SAW requirements stored in Situation Data Manager, and notifies other components of the recognized situations. We have developed a new situation analysis process, which eliminates redundant computation and allows runtime modifications of SAW requirements.
- *Action Scheduler* providing R3. It identifies the actions to be triggered by the recognized situations, invokes the security policy enforcement component to check whether the actions are permitted under the situation, and schedule the permitted actions.
- *SA Communication Subsystem* providing R4. It discovers a suitable remote object to communicate with the local object based on the recognized situations, and establishes the communication channel between the objects. For this component, we reuse the design for our RCSM.

In addition, Situation Data Manager in our SA middleware stores the SAW requirements of trustworthy ubicomp application software, which are needed by other components in our SA middleware to perform their tasks. Situation Data Manager also maintains the relevant context and situation history for analyzing complex situations. By sharing context and situation information among multiple trustworthy ubicomp application software, Situation Data Manager also improves the efficiency of our SA middleware. The security policy enforcement component in our SA middleware for trustworthy ubicomp application software will be discussed in Section 8.

6.2 Components of the SA Middleware

Due to the severe resource constraints of ubicomp devices and the dynamicity of ubicomp environments and software requirements, an SA middleware must satisfy the following requirements:

M1. It must adapt to the needs of trustworthy ubicomp application software.

M2. It must be lightweight and energy-efficient.

M3. It must operate in a timely manner.

Therefore, the components in the SA middleware must satisfy M1-M3. In this subsection, we will discuss each major component of the SA middleware, except Action Scheduler. Further investigation is needed to identify the most appropriate scheduler for trustworthy ubicomp application software among existing schedulers [23, 24].

- *Context Manager*

Context Manager must have the following capabilities to support transparent and on-demand context acquisition:

CM1. Local sensing unit management, which includes maintaining the information of sensing units on the device, and dynamic addition, removal or reconfiguration of local sensing units.

CM2. Context acquisition, which includes acquiring context data from local sensing units and responding to context acquisition requests from local Situation Data Manager or from remote Context Managers.

CM3. Context discovery, which includes discovering remote sensing units dynamically through collaboration with remote Context Managers when the requested context data is not available in the local device.

In the above capabilities CM1-CM3, the context discovery and context acquisition from remote Context Managers require communication among devices, which is likely to cause long delay and consume much energy. Hence, the key issue for developing Context Manager is the development of a protocol for context discovery and remote context acquisition, which satisfies M1-M3. We have developed an adaptive, lightweight and energy-efficient context discovery protocol, R-CDP [19], to address this issue. A Context Manager has been developed using R-CDP (see Section 9). Other protocols or mechanisms may also be used to develop Context Manager if they satisfy M1-M3 and provide the required capabilities CM1-CM3.

- *Situation Analyzer*

Situation Analyzer analyzes the relevant context and situation history maintained by Situation Data Manager to determine the current situation based on SAW requirements of the trustworthy ubicomp application software. When contexts are updated by Context Manager, Situation Analyzer starts a new round of analysis of each situation of interest to the application software. The analysis process in Situation Analyzer should have the following characteristics to satisfy M1-M3:

C1. Any situation should be evaluated at most once in each round. When a situation is used to trigger actions of multiple trustworthy ubicomp application software or to compose several more complex situations, Situation Analyzer should only evaluate this situation once in each round and make the result available for further usage.

C2. When the SAW requirements of a trustworthy ubicomp application software change, Situation Analyzer should automatically start to analyze the new situations based on the new requirements without off-line reconfiguration.

Based on our SAW ontology, we have established the following situation analysis process with the two characteristics C1-C2:

Step 1. Whenever updates on contexts $\{c_0, c_1, \ldots, c_k\}$ are collected by Context Manager, Situation Analyzer starts a new round of situation analysis.

Step 2. For each atomic situation a_situ_i, if a_situ_i is not defined using any context in $\{c_0, c_1, \ldots, c_k\}$, mark a_situ_i as "no change" and continue to evaluate the next atomic situation. Otherwise, evaluate a_situ_i as follows:

Step 2.1 Get the value of each argument of a_situ_i by retrieving the corresponding context values from the context history maintained by Situation Data Manager.

Step 2.2 Calculate the value of a_situ_i by applying the context operators used in a_situ_i on the corresponding argument values retrieved in Step 2.1.

Step 2.3 Store the value of a_situ_i in Situation Data Manager.

Step 3. For each composite situation c_situ_m, if all the situations composing c_situ_m are marked as "no change", mark c_situ_m as "no change", and evaluate the next composite situation. Otherwise, evaluate c_situ_m as follows:

Step 3.1 Retrieve the value of each situation composing c_situ_m from Situation Data Manager.

Step 3.2 Calculate the value of c_situ_m by applying the logical or temporal operators composing c_situ_m on the values of atomic situations retrieved in Step 3.1.

Step 3.3 Store the value of c_situ_m in Situation Data Manager.

- *SA Communication Subsystem*

To support efficient SA communication, SA Communication Subsystem must have the following capabilities:

SAC1. Peer-to-peer dynamic discovery of remote SA objects that are suitable for communicating with a local SA object which needs the communication under the current situation.

SAC2. Establishment of SA communication channel between two SA objects over wireless or wired networks.

SAC3. Management of local SA objects that require communication with remote SA objects.

Capability SAC1 is required because the device mobility and the lack of infrastructure support in ubicomp environments make infrastructure-based approaches [25-27] inappropriate. Capability SAC3 is required for providing the necessary information of SA objects for object discovery in SAC1.

We have developed adaptive energy-efficient protocols to support SA communication in ubicomp environments [3, 20, 21], and used these protocols to develop an SA Communication Subsystem (see Section 9). Other protocols or mechanisms may be used to develop SA Communication Subsystem if they satisfy M1-M3 and provide the capabilities SAC1-SAC3.

7. Providing Development and Runtime Support for SAW in Trustworthy Ubicomp Application Software

Our approach to providing development support D1, D2 and D4, and the runtime support R1–R4 for SAW in trustworthy ubicomp application software consists of the following steps:

S1. Develop a language for specifying SAW requirements D1 and D2 based on our SAW ontology.

S2. Use an SA middleware to provide runtime support R1-R4 for SAW.

S3. Design an architecture of trustworthy ubicomp application software, which will be running on top of the SA middleware, and develop a tool for code generation D4 based on the SAW requirement specifications.

We will discuss each of these three steps in the following subsections.

7.1 An SAW Requirement Specification Language for Trustworthy Ubicomp Application Software Development

Based on our SAW ontology, we have developed Situation-Aware Interface Definition Language (SA-IDL) for application developers to specify SAW requirements of trustworthy ubicomp application software. The specifications will serve as the basis of runtime situation analysis and automated code generation for the software. To achieve concise and reusable specifications, object-oriented representation for contexts is used in SA-IDL. Since modifying the definition of a context class will automatically affect its subclasses, trustworthy ubicomp application software can be easily modified by either changing the existing context specifications or adding new context specifications.

As shown in the Table 1, SA-IDL provides the necessary primitives for specifying SAW requirements based on our SAW ontology:

Table 1. Primitives in SA-IDL for the entities in our SAW ontology

Entities in our SAW Ontology	*Primitives in SA-IDL*
context	class
context instance	variable representing a class object in operand
location identifier	Location class (SA-IDL reserved)
time identifier	Time class (SA-IDL reserved)
argument	operand
context instance	an instance of a context class

context operator	function, arithmetic operator, compare operator, time constraint
atomic situation	atomic situation
composite situation	composite situation
situation operator	logical operator, time constraint
action	action
situation triggers action relationship	rule

1) <attribute> ::= <attribute type> <attribute name> ';'
2) <attributes> ::= <attribute> [<attribute> …]
3) <class> ::= 'Class' <class name> ['extends' <parent class>] '{' <attributes> '}'
4) <function> ::= 'AVG' | 'MAX'| 'MIN' |'DELTA' | 'numOfList'
5) <arithmetic operator> ::= '+' | '-' | '*' | '/' | 'IN'
6) <logical operator> ::= '&&' | '||' | '!'
7) <compare operator> ::= '>' | '>=' | '<' | '<=' | '= =' | '!='
8) <parameter> ::= <parameter type> <var name>
9) <parameters> ::= <parameter> [',' <parameter> …]
10) <operand> ::= <variable name> '.' <attribute name> | <a time object> | <a location object> |
<a string> | <a integer> | <a real number> | <a Boolean constant> | 'actionSignature'
11) <time constraint> ::= '[' 'forAny' | 'exist' ',' <time stamp> ',' <offset> ',' <interval> ']'
12) <situation> ::= <atomic situaton> | <composite situation>
13) <atomic situation> ::= 'AtomicSituation' <situation name> '(' <parameters> ')'
[<function>] <operand> [<arithmetic operator> <operand>] <compare operator> <operand>
14) <composite situation> ::= 'CompositeSituation' <situation name> '('<parameters> ')'
[<time constraint>] <atomic situation>
| <atomic situation> <logical operator> <atomic situation>
15) <action> ::= <return type> <action name> '(' <parameters> ')'
16) rule ::= 'Rule' <rule name> 'ActiveAt' <situation> '{' '[' 'local' | 'incoming' | 'outgoing' ']' <action> ';' … … '}'

Figure 4. All the productions in SA-IDL grammar.

We use the Backus-Naur form (BNF) to specify the grammar of our SA-IDL. Figure 4 shows all the productions in SA-IDL's grammar. The left-hand sides of the productions are non-terminal symbols, which are enclosed in <>. All terminal symbols are enclosed in ' '. The items quoted in [ ] are optional.

7.2 Using SA Middleware to Provide Runtime Support

The following process shows how SA middleware facilitates the execution of trustworthy ubicomp application:

P1. **Application registration**: When a trustworthy ubicomp application software starts running, it sends a registration message to the SA middleware. The registration message contains the interfaces of SA objects in the software and the SAW requirement specifications of the software. Once the SA middleware receives a registration message, Situation Data Manager in the SA middleware parses the message, stores new object information and SAW requirement specifications, and notifies other components of the new SAW requirements.

P2. **Middleware reconfiguration**: Upon receiving the specifications of new SAW requirements, Context Manager locates the necessary context sources and starts collecting the new contexts. Situation Analyzer starts analyzing situations defined in the specifications. Similarly, Action Scheduler and SA Communication Subsystem start triggering and scheduling actions, and establishing communication channels based on the specification.

P3. **Context acquisition (R1)**. Context Manager periodically acquires context updates from local sensing units, and/or listens to context update messages from remote SA middleware instances. Whenever some contexts are updated, Context Manager adds the context updates into the context history maintained by Situation Data Manager. Context Manager also listens to context request messages from remote SA middleware instances, and sends the requested contexts to the remote SA middleware instances if the requested contexts are available.

P4. **Situation analysis (R2)**. When new context updates are available in Situation Data Manager, Situation Analyzer analyzes the context and situation history in Situation Data Manager to recognize the situations based on the SAW requirements. If a situation is recognized, Situation Analyzer notifies Situation Data Manager, Action Scheduler, SA Communication Subsystem and security policy enforcement component.

P5. **Action triggering and scheduling (R3)**. When Action Scheduler receives a recognized situation from Situation Analyzer, it triggers proper actions based on the specifications of SAW requirements. When multiple actions need to be triggered, Action Scheduler generates a schedule of these actions based on their real-time requirements, and triggers the actions according to the generated schedule.

P6. **SA communication (R4)**. If a triggered action of an SA object involves inter-object communication, SA Communication Subsystem performs SA object discovery to locate a remote object to communicate with and establish the communication channel between these two objects.

7.3 Architecture and Code Generation for Trustworthy Ubicomp Application Software

After specifying the SAW requirements of trustworthy ubicomp application software using SA-IDL, it is desirable to have a tool that can automatically generate codes of the application software based on the specification (**D4**). To develop such a tool, an architecture of trustworthy ubicomp application software must be developed first. With an SA middleware that provides the required runtime support, the design of trustworthy ubicomp application software is greatly simplified in the following ways: Only the functional components of the application software and the interfaces with the SA middleware need to be developed. For the same SA middleware, the interfaces can be reused by various trustworthy ubicomp application software using the same SA middleware. As discussed in Sections 6 and 7.2, a trustworthy ubicomp application software needs to send its SAW requirements to the SA middleware to properly reconfigure the SA middleware. Therefore, a standard data format needs to be defined for the SA middleware to understand the SAW requirements of the trustworthy ubicomp application software.

Based on the above observations, we develop an architecture of trustworthy ubicomp application software as shown in Figure 5. In this architecture, application skeletons integrate the functional components of the software with the interfaces with an SA middleware. An application skeleton has a base class, an App comm. and an SA file. The base class is a class with abstract methods, which are from the action signatures in SA-IDL specifications. Since actions are application-dependent, they should be implemented by application developers in an application object that extends the base class. The SAW requirements are specified in an SA file, which is an XML file containing the specifications for contexts, context collection rates, situations and rules. App comm. is the communication interface between the software and an SA middleware, and hence is middleware-dependent. Based on the execution process of trustworthy ubicomp application software in Section 7.2, App comm. processes the following three types of messages between application software and an SA middleware: a) application registration/deregistration, b) action triggering, and c) action results.

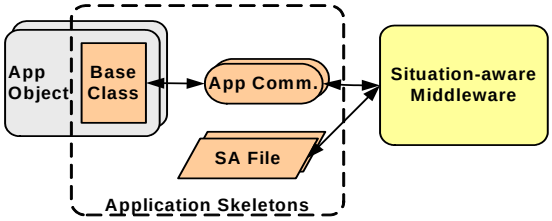

Figure 5. An architecture of trustworthy ubicomp application software

In this architecture, application developers only need to focus on the application-dependent functionality, which is the implementation of the abstract methods in application objects. The SA middleware takes care of SA communication, context discovery and acquisition, data management, situation analysis, and action triggering and scheduling for multiple trustworthy ubicomp application software running on the same device simultaneously. Such a design greatly enhances information sharing and code reusability, and hence reduces the memory consumption for data management, as well as CPU usage for situation analysis and communication overhead for context acquisition.

Assuming a particular SA middleware is selected to provide runtime support and a specific programming language is selected for the software development, the development process of an automated code generation tool for trustworthy ubicomp application software based on this architecture can be described as follows:

i. Develop an SA-IDL parser for checking syntax and parsing SA-IDL specifications using existing parser construction tools, such as Lex and Yacc.
ii. Determine the format of the SA file based on the selected SA middleware.
iii. Develop a standard library for the interfaces with the selected SA middleware.

iv. Develop a code generator that takes the parsing result of an SA-IDL specification using the SA-IDL parser developed in (i) as input, maps the action signatures in SA-IDL specifications into class definitions in the selected programming language, converts the SAW requirements into the SA file with the format defined in (ii), and incorporates the standard library developed in (iii) as App Comm.

In the above process, the resulting SA-IDL parser from (i) can be reused for developing various code generators for different SA middleware and programming languages. Once a code generator is developed, the application skeleton of trustworthy ubicomp application software can be automatically generated by parsing SA-IDL specification of the applications software.

8. SA Security Policy Specification and Enforcement in Trustworthy Ubicomp Application Software

With our development and runtime support, SA security policies can be specified and enforced in trustworthy ubicomp application software. SA security policies are rules defining security requirements, such as requirements on authentication and authorization, under various situations. By incorporating SAW in the process of security policy specification and enforcement, we can dynamically change the privileges of subjects in the software under various situations such that at runtime only minimal privileges (or permissions) are granted to subjects as needed under current situation. Hence, the attacks of misusing privileges can be reduced, and the related objects can be properly protected under various situations.

8.1 A core ontology for modeling SA security policies in trustworthy ubicomp application software

To specify security policies in trustworthy ubicomp application software, we have developed a core ontology of SA security policies. An ontology of security policies is a representation of a specific concept model that describes complex relationships among the entities involved in security policies. The core ontology we have developed incorporates a general and extensible model for representing SA security requirements of multiple entities in trustworthy ubicomp application software. Figure 6 shows the following generic classes in the core ontology:

- *Entity*: a generic security entity class for policy specification.
- *Subject*: a generic class for the subject of an action.
- *Object*: a generic class for the application objects in trustworthy

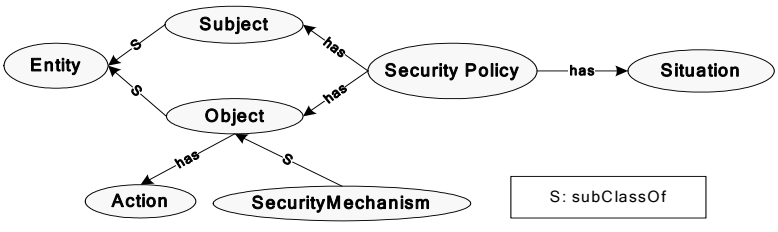

Figure 6. The core ontology for modeling security policies in trustworthy ubicomp application software

ubicomp application software. An object may perform several actions.

- *Action*: a class for the requested actions which invoke a method of an object with certain parameters.
- *Situation*: a generic class for situations used in the security policies. The situation is specified and recognized using our development and runtime support.
- *SecurityMechanim*: a generic class for specifying which security mechanism should be used. Different security mechanisms, e.g. plaintext authentication, windows domain authentication and SSL/TSL authentication, often imply different levels of security enforced. As security is difficult to be quantified, different security levels are often represented using different security mechanisms. Hence, security policies often involve what security mechanisms to be used under various situations.

With this core ontology, the following three types of SA security policies in trustworthy ubicomp application software can be specified:

- Authentication:

(SP1) A security mechanism ***m*** must be used for authenticating a subject ***p*** under the situation ***s***.

(SP2) A subject ***p*** is trusted under situation ***s*** if ***p*** has been authenticated using the security mechanism ***m***.

- Access control:

(SP3) A trusted subject ***p*** can perform an action α of an object ***o*** under the situation ***s***.

Note that SP1 states a requirement for the security mechanism. SP2 states the required condition for p to be trusted under situation S. There are other types of security policies that cannot be specified using the current SA-IDL, such as provisional authorization policies and delegation policies. We are currently extending the current SA-IDL to support the specification of other security policies.

8.2 Specifying SA security policies using SA-IDL

In this section we will present a method for specifying security policy types SP1 – SP3 using SA-IDL, in which entities in the core ontology are specified using the "Class" in SA-IDL and security policies are specified using the "Rule" and "Situation" in SA-IDL (see Section 7.1):

SPEC1. Define the classes which are used in SP1-SP3. In addition to the classes defined in the core ontology, define the following class to specify SP2:

Class *ActionHistory* {ListAction actionPerformed;} //The class definition for a context *ActionHistory*

SPEC2. Define the situations which are used in SP1-SP3. In addition to the situation ***s*** used in SP1-SP3, define the following two additional situations:

- authenRequestReceived(Subject p), which indicates that a request for authenticating a subject ***p*** was received.
- triggerRequestReceived(Action a), which indicates that a request for triggereing an action ***a*** was received.

If the security policy enforcement component can provide the information on the received requests as context data, the above two situations can be recognized by Situation Analyzer in the SA middleware.

SPEC3. A security policy of type SP1 can be defined as follows:

```
Subject              p;
SecurityMechanism    m;

CompositeSituation authenUnderS (Subject p)
        authenRequestReceived(p) && s;      //Indicates that p needs to be authenticated under s
Rule authenticate ActivateAt authenUnderS { [local] bool callSecMechanism(m, "authenticate", p) } ;
```

In the above specification, we assume that the security policy enforcement component in the SA middleware will provide a ***callSecMechanism*** action, which will actually use ***m*** to authenticate ***p*** when ***callSecMechanism***(***m***, "authenticate", ***p***) is triggered. The rule in the above specification says that when ***p*** needs to be authenticated under ***s***, a ***callSecMechanism*** action should be triggered using the security mechanism ***m***, the subject ***p*** and a string "authenticate" as parameters. Under our assumption on the ***callSecMechanism*** action, the rule in the above specification is equivalent to the security policy of type SP1, and can be correctly executed in runtime to enforce the policy.

SPEC4. A security policy of type SP2 can be defined as follows:

```
CompositeSituation trusted (Subject p, SecurityMechanism m, ActionHistory h)
        [exists, TS_NOW, null, null] (x IN h.actionPerformed
                        && "callSecMechanism" == x.name     //an action CallSeMechanism is performed
                        && m IN x.params && "authenticate" IN x.params
                        && p IN x.params                    //the performed action is to authenticate p
                        && s)                               //the trust is active only when s is true
```

As mentioned in Section 7, "[exists, TS_NOW, null, null]" is a time constraint defined in SA-IDL. TS_NOW is a timestamp representing the current time. "exists" is the existential quantifier over a certain time period [TS_NOW – offset, TS_NOW – offset + interval], where the offset is the difference between the beginning of the time period and TS_NOW, and the interval is the length of the time period. Since the offset and interval of the time period are not specified in "[exists, TS_NOW, null, null]", the time constraint "[exists, TS_NOW, null,

null]" means sometime in the past. The definition of the above composite situation indicates that a callSecMechanism action, which used ***m*** to authenticate ***p***, has happened. According to the definition of security policies of type SP2, when ***s*** is true, ***p*** is trusted. Therefore, the situation *trusted* specifies a security policy of type SP2.

SPEC5. Using the situation *trusted* defined in SPEC4, a security policy of type SP3 can be defined as follows:

```
Subject             p;
SecurityMechanism   m;
Action              α;
ActionHistory       h;

CompositeSituation actionPermitted (Subject p, Action α, SecurityMechanism m, ActionHistory h)
        trusted(p, m, h) && triggerRequestReceived(α) && s;

Rule trigger ActivateAt actionPermitted(p, α, m, h) { […] void a (…) };
```

Based on the definition of the composite situation *actionPermitted*, it is true when a request for triggering an action ***α*** was received, ***p*** is trusted, and ***s*** is true. Therefore, the rule in the above specification defines a security policy of type SP3.

8.3 Enforcing SA security policies in trustworthy ubicomp application software using SA middleware

To enforce SA security policies in trustworthy ubicomp application software, a security policy enforcement component is developed in the SA middleware as shown in Figure 3. This security policy enforcement component has the interfaces, such as the *callSecMechanism* action in the specification of a security policy of type SP1, to a set of routines that implement various security mechanisms. During runtime, this component works with other components (Situation Data Manager, Context Manager, Situation Analyzer and Action Scheduler) in the SA middleware to enforce SA security policies:

E1. Situation Data Manager receives the registration message of the trustworthy ubicomp application software, parses the SA File to extract the SAW requirements and security policies of the application software, and sends the context requirements to Context Manager.

E2. Context Manager collects the required contexts and stores them to Situation Data Manager

E3. Situation Analyzer analyzes the context and situation history maintained by Situation Data Manager, and notify Action Scheduler and the security policy enforcement component of the current situation.

E4. Action Scheduler notifies the security policy enforcement component of the actions to be triggered under the current situation.

E5. The security policy enforcement component retrieves the security policies related to the actions to be triggered from Situation Data Manager, evaluates the security policies to determine whether the actions are allowed and/or what security mechanisms need to be used to authenticate the subjects/objects before performing these actions, and invokes the appropriate routines if necessary.

E6. Based on the results of Step E5), the security policy enforcement component notifies Action Scheduler of the actions that are allowed and the actions that are denied.

E7. Action Scheduler schedules and invokes the actions allowed, and notifies the application software of the actions that are denied.

9. Implementation and Evaluation of Our Approach

Our new design of SA middleware (Section 6) keeps the main features of our RCSM [1, 20, 28] mentioned in Section 4, and overcomes the difficulties of the RCSM. Based on this design, we have improved RCSM to provide development and runtime support for trustworthy ubicomp application software. Figure 7 shows the architecture of the improved RCSM. Situation-aware Processor (SA Processor), which is application independent and can support multiple trustworthy ubicomp application software simultaneously, implements Situation Data Manager, Situation Analyzer, and Action Scheduler. In particular, Situation Analyzer implements the situation analysis process in Section 6.2, which effectively eliminates redundant computation in situation analysis. Action Scheduler implements a simple priority-based scheduling algorithm. The improved RCSM Object Request Broker (R-ORB) implements Context Manager described in Section 6.2, which supports context discovery, in addition to its original SA Communication Subsystem. In addition, a Situation-Awareness specification Management tool (SAM tool) has been developed for the improved RCSM to allow users to modify SAW requirements in runtime. We have implemented a new SA-IDL compiler, which can generate application skeletons for trustworthy ubicomp application software running on top of the improved RCSM. Application skeletons, SA Processor, R-ORB and the SAM tool are implemented in eMbedded VC++ for Windows CE platforms.

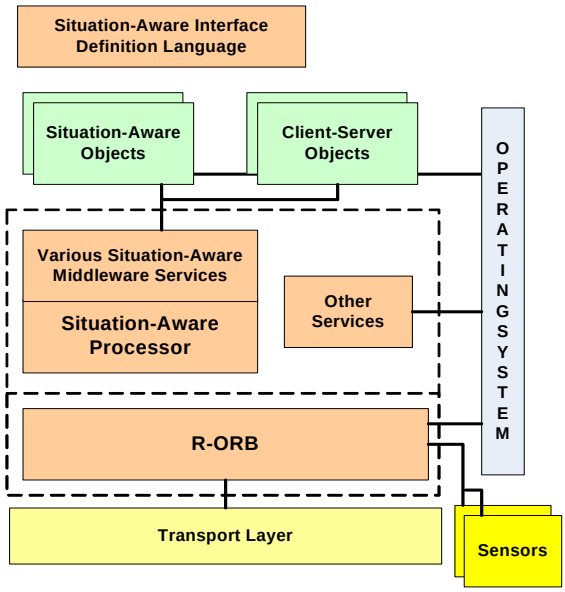

Figure 7. Improved RCSM architecture.

Our evaluation of the improved RCSM mainly focuses on the CPU usage, memory load, energy consumption and latency caused by RCSM. The energy consumption and latency caused by RCSM mainly attribute to wireless communication, which occurs in SA communication and context discovery. In [3, 21], the evaluation results of our SA communication protocols used in R-ORB have been presented. The results in [3, 21] show that the added latency of SA communication compared to client-server communication is negligible (milliseconds) even when there are more than 100 application objects with 10 methods or more in each object, and our object discovery protocol [3] consumes less energy than other broadcast/flooding-based protocols considering the total energy consumption of the entire network. In [19], the evaluation results of our context discovery protocol used in R-ORB have been presented. The results in [19] show that the latency of context discovery is mainly determined by the end-to-end latency in wireless communication. The results in [19] also show that as the number of nodes (context providers and requesters) in the network increases, the energy consumption of our context discovery protocol only shows marginal increase by adaptively reducing the number of beacons for context discovery request and result transmission.

To obtain more complete results for evaluating the CPU usage, memory load, energy consumption and latency caused by RCSM, we have performed the following experiments using DELL AximX5 PDAs (with Intel® XScale™ 400MHz processor, 64MB RAM and 48MB ROM, DELL TrueMobile™ 1180 WLAN CompactFlash Card, and 1440mAH Li-ion battery). We selected DELL AximX5 PDAs because most devices based on Windows CE platforms, such as PDAs, smart phones, wearable PCs, and vehicle-mounted computers, have similar or even better hardware configurations.

(a) The experiment for evaluating energy consumption and latency caused by RCSM.

We developed an application to run on RCSM for our experiment. The application requests five contexts: time, location, noise, light, and neighbor list. Four PDAs were used in this experiment. One PDA serves as the context provider, which periodically broadcasts (in 1 second interval) the location, time, noise and light. RCSM running on each PDA periodically broadcasts (in 1 second interval) the user information of the PDA to maintain the neighbor list. The context provider sends out a special combination of the contexts every eight minutes. The SA Processor in RCSM on each PDA recognizes this as a situation and triggers an action of the application. To perform the triggered action, this application finds another application running on a neighbor PDA and triggers a method to transmit 1 MB data to the application on the neighbor PDA. We have also added codes in RCSM to record the remaining battery level every eight minutes, the end-to-end latency (from recognizing the situation to finishing the data transmission)

and the internal processing latency of RCSM (without the latency caused by data transmission) for each situation-triggered action. To find out the energy consumption caused by RCSM, we manually recorded the energy consumption of an idle PDA (with its network adapter turned on).

The results show that the average end-to-end latency for the triggered action in our experiment is 10.7 seconds with 1MB data transmission, and the average internal processing latency is 0.29 second. Hence, the average latency caused by data transmission, which is the difference between the average end-to-end latency and the average internal processing latency, is about 10.4 seconds. From these results, we can see that the average internal processing latency of RCSM is negligible compared to the latency caused by data transmission, which is mainly determined by underlying wireless network protocols that are not under control of RCSM. Therefore, the latency caused by RCSM is acceptable.

The results also show that the average energy consumption of the PDAs used in the experiment, except the one serving as the context provider, is 23% of the battery power in 2 hours, whereas the idle PDA consumes 15% of the battery power in 2 hours. Hence, in our experiment, the average energy consumption caused by RCSM with a running application is 8% of the battery power in 2 hours, which is quite reasonable considering that this energy consumption includes the energy used for transmitting useful application data, broadcasting user information every second, and receiving context updates from the context provider every second. In actual applications where user information and contexts are not broadcasted/updated so frequently, the average energy consumption caused by RCSM is expected to be much less than the average energy consumption caused by RCSM in our experiment.

(b) The experiment for evaluating the CPU usage and memory load caused by RCSM.

For our experiments, we developed ten applications, each of which has five actions triggered by five different situations. We run these experimental applications on the same device, and used the Remote Performance Monitor, which is integrated in Microsoft eMbedded Visual C++ 4.0, to collect the CPU and memory usage information. The results show that after RCSM and one experiment application starts running, the memory load of the PDA increases 4%, and the average total processor time usage increases only 0.6%. When we increase the number of applications running on RCSM from one to ten, the memory load of the PDA increases another 9%, and the average total processor time usage increases another 1%. These results show that RCSM is lightweight and scalable.

10. Revisit the Example

In this section, we will illustrate our approach using the example presented in Section 2. In this example, we assume that three sensing units are provided in the smart classroom: (1) an *RFID sensor* to detect RFID signals and retrieve personal information stored in RFID tags, (2) a *classroom monitor* to keep track with different RFID signals and the number of people in the classroom, and (3) a *course monitor* to retrieve course information from a database.

10.1 Requirement analysis and SA-IDL specification of Trustworthy Ubicomp Application Software

The SAW requirements of the trustworthy ubicomp application software in our example are analyzed using the process presented in Section 5.

Step (1): Identify the following actions and situations triggering these actions.

- Action "openDoor" is triggered when "a person stands in front of the door for 3 seconds" (*atDoorForAWhile*) and "the person is allowed to enter the classroom" (*allowToEnter*). The situation *allowToEnter* specifies the access control policy of the smart classroom described in Section 2.
- Action "alarm" is triggered when "someone enters the classroom without proper authorization" (*intruderInClassroom*). The situation *intruderInClassroom* specifies how a violation of the access control policy of the smart classroom is captured.

Step (2): Decompose the identified situations to atomic situations.

- *atDoorForAWhile* is decomposed to an atomic situation "a person stands in front of the door" (*atDoor*) and a temporal operation on this atomic situation "a situation is true for 3 seconds".
- *allowToEnter* is the logical disjunction of the following situations:
  - "An instructor is allowed to enter the classroom" (*allowInstructorToEnter*)
  - "A student is allowed to enter the classroom" (*allowStudentToEnter*)
- *intruderInClassroom* is the logical disjunction of the following situations:
  - "The number of persons entering the classroom is greater than the number of different RFID signals in the classroom" (*unidentifiedPersonInRoom*)
  - "A person in the classroom is not allowed to enter the classroom" (*enterWithoutPermission*)
- Repeat Step (2) until no further decomposition can be made. The resulting decomposition is listed as follows:
  - $allowInstructorToEnter = withValidID \wedge isInstructor \wedge timeForClass$
  - $allowStudentToEnter = withValidID \wedge isStudent \wedge timeForClass$

- *enterWithoutPermission = inClassroom* $\wedge \neg$ *allowToEnter*

Step (3): Identify the relevant contexts and operations on the contexts.

Tables 2 and 3 show the relevant contexts based on the situations identified in Step (2), as well as the operations on these contexts. Figure 8 shows a portion of the SA-IDL specification for the application software, which is named "DoorGuard", based on the identified SAW requirements using Step (1) – (3).

Table 2. The relevant contexts in the situations identified in Step (2)

Contexts / ***Situations***		***Time***	***Course Information***	***Person Information***	***Classroom Information***
atDoorForAWhile	*atDoor*	✓		✓	✓
allowToEnter	*isInstructor*	✓		✓	
	isStudent	✓		✓	
	timeForClass	✓	✓		
	withValidID	✓	✓	✓	
intruderInClassroom	*unidentifiedPersonInRoom*	✓			✓
	inClassroom	✓		✓	✓
	allowToEnter	✓	✓	✓	✓

Table 3. The operations on the contexts in Table 2.

Contexts	***Properties of Contexts***	***Operations on Contexts***
Time	*Year, month, day, weekday, hour, minute, second*	*=, !=, >=, <=*
Location	*String identifying a location*	*=, !=*
Course Information	*Start time, end time, list of IDs*	*operations on time and IN (determine one element is in a list of elements) on list of IDs*
Person Information	*ID, current location and role*	*operations on time and location*
Classroom Information	*Door location, number of people inside, list of ids*	*operations on location and integer and IN on list of IDs*

10.2 Code generation for Trustworthy Ubicomp Application Software using the SA-IDL compiler

Using the SA-IDL compiler, the application skeleton for *DoorGuard* can be generated from its SA-IDL specifications. The application skeleton of *DoorGuard* can be divided into three parts: (1) a base class with two abstract methods "*void openDoor()*" and "*void alarm()*", (2) the communication interfaces with R-ORB and SA Processor, and (3) an SA file *DoorGuard_SAInfo.xml* containing the SAW requirement specifications of *DoorGuard*. Figure 9 shows some parts of the generated SA file. To complete the development of *DoorGuard*, the developers only need to extend the base class by implementing the two abstract methods in *DoorGuard*.

10.3 Execution of *DoorGuard*

```
RCSMApplication DoorGuard {
  // Time and Location context classes are system defined.
  Class Time {int year; int month; int day;
              int hour; int minute; int second; int weekday;}
  Class Location {String loc;}
  Class Course {Time beginTime; Time endTime; List string idList;}
  Class Classroom {Location doorLoc; List String idInRoom;  int numOfPeople; }
  Class Person {String ID; Location loc; String role;}
  Context Collection Rate = 1;
  AtomicSituation unidentifiedPersonInRoom (Classroom clsm)
      clsm.numOfPeople > numOfList(clsm.idInRoom);
  … …
  CompositeSituation atDoorForAWhile(Person p, Classroom clsm)
      [ForAny, TS_NOW, 3, 3]atDoor(p, clsm);
  CompositeSituation allowStudentToEnter (Person p, Classroom clsm, Course crs)
      timeForClass(crs) && isStudent(p) && withValidID(p, crs);
  … …
  CompositeSituation allowToEnter (Person p, Person instructor, Classroom clsm,
                                   Course crs)
      allowStudentToEnter(instructor, p, clsm, crs)
    || allowInstroctorToEnter(p, clsm, crs);
  CompositeSituation intruderInClassroom (Person p, Person instructor,
                                          Classroom clsm, Course crs)
      unidentifiedPersonInRoom(clsm)
    || (inClassroom(p, clsm) && ! allowToEnter(p, instructor, clsm, crs))
  CompositeSituation readyToOpenDoor (Person p, Person instructor,
                                      Classroom clsm, Course crs)
      atDoorForAWhile(p, clsm) && allowToEnter(p, instructor, clsm, crs);
  Rule openDoor ActivateAt readyToOpenDoor(Person p, Person instructor,
                                           Classroom clsm, Course crs) {
      [local] void openDoor()} ;
  Rule alarm ActivateAt intruderInClassroom (Person p, Person instructor,
                                             Classroom clsm, Course crs) {
      [local] void alarm()};
}
```

Figure 8. Partial SA-IDL specification of DoorGuard

```
- <SA application="DoorGuard">
  - <ClassList rate="1">
    + <Class name="Course">
    - <Class name="Person">
        <attribute name="ID" type="string" />
        <attribute name="loc" type="Location" />
        <attribute name="role" type="string" />
      </Class>
    </ClassList>
  - <AtomicSituationList>
    + <AtomicSituation name="isInstructor" parameters="Person p">
    </AtomicSituationList>
  - <CompositeSituationList>
    - <CompositeSituation name="atDoorForAWhile" parameters="…"
        quantifier="forAny" ts="TS_NOW" offset="3" interval="3">
        <root value="inFrontOfDoor" parameters="…" />
      </CompositeSituation>
    + <CompositeSituation name="allowToEnter" parameters="…">
    - <CompositeSituation name="readyToOpenDoor" parameters="…">
      - <root value="&&">
          <lhs value="atDoorForAWhile" parameters="…" />
          <rhs value="allowToEnter" parameters="…" />
        </root>
      </CompositeSituation>
    </CompositeSituationList>
  - <RuleList>
    - <rule name="openDoor">
        <situation name="readyToOpenDoor" parameters="… …" />
      - <action type="local">
          <name>openDoor</name>
          <returnType>void</returnType>
          <parameters />
        </action>
      </rule>
    + <rule name="alarm">
    </RuleList>
  </SA>
```

Figure 9. Partial SA file generated from SA-IDL specifications of DoorGuard.

Now we will illustrate how RCSM supports the execution of *DoorGuard*. Assume that the smart classroom is Room B468, and a class *Chemistry330* is held in this room from 10:40 a.m. to 11:55 a.m. on every Tuesday. The instructor of *Chemistry330* is Professor Smith.

Before running *DoorGuard*, the three sensing units in Room B468 registers to R-ORB running on the same device with *DoorGuard*. Once *DoorGuard* starts running, it registers to SA Processor, which extracts the SAW requirements of *DoorGuard* from *DoorGuard_SAInfo.xml*, and notifies R-ORB of the three contexts to be collected. R-ORB finds all three contexts from the registered sensing units, notifies SA Processor the availability of these three contexts and activates the three sensing units to collect contexts. This process illustrates the on-demand context acquisition support provided by R-ORB.

Suppose the current time is 10:30 a.m., September 25, 2005, which is ten minutes before the starting time of *Chemistry330*. Professor Smith comes to the door of Room B468 and stands there for 3 seconds with Mike, who is a visitor without any RFID

tag. R-ORB collects the personal information of Professor Smith, the class information of *Chemistry330* and the classroom information through the three sensing units, and sends the collected data to SA Processor. SA Processor generates the result that the situations *allowToEnter* and *atDoorForAWhile* are both true for Professor Smith, and hence triggers the *openDoor* action to let Professor Smith enter Room B468. This process illustrates the enforcement of an access control policy using R-ORB and SA Processor.

Suppose Mike also enters the classroom when the door is opened for Professor Smith. Since Mike does not carry any RFID tag, the classroom monitor detected two persons entering the classroom, but only one RFID signal. Based on this information, SA Processor generates the result that the situation *intruderInClassroom* is true, and triggers the *alarm* action to warn Professor Smith that an unauthorized person is in the classroom. This process illustrates the detection of a violation of the access control policy to the smart classroom.

10.4 Reconfiguration of *DoorGuard*

Suppose *DoorGuard* is already running. Before an upcoming *Chemistry330* class, Professor Smith wants to change the original plan of giving a lecture to instructing students to do some experiments. Since inappropriate operations on the chemicals are very dangerous, students cannot be left alone in the classroom. Thus, a new requirement "a student can enter the classroom only when the instructor is inside" is added. The composite situation "*allowStudentToEnter*" should be reconfigured by concatenating a new atomic situation "*instructorInClassroom*"

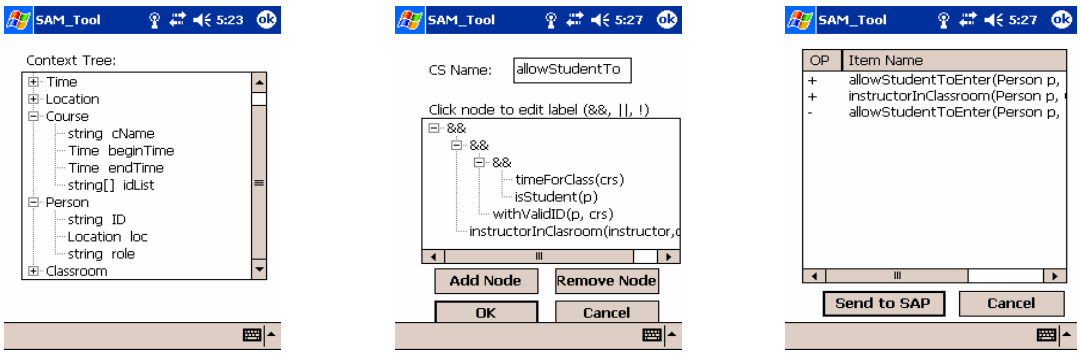

Figure 10. GUIs of the SAM tool. From left to right: The GUI for displaying context definitions, modifying situation definitions, and sending updated requirement specifications to the SA Processor

with a conjunction operator. The new atomic situation and the reconfigured composite situation are given below:

AtomicSitucation instructorInClassroom(Person p, Classroom clsm)
p.id IN clsm.idInRoom;
CompositeSituation allowStudentToEnter (Person p, Person inst, Classroom clsm, Course crs)
timeForClass(crs) && isStudent(p) && withValidID(p, crs) && instructorInClassroom(inst,clsm);

To incorporate the new requirement into *DoorGuard*, Professor Smith needs to use the SAM tool (see Section 9) to reconfigure SA Processor. The GUIs of the SAM tool are shown in Figure 10. Once the new requirement is

submitted to SA Processor through the SAM tool, *DoorGuard* starts enforcing the new policy. There is no need to change any codes, or terminate the application for updating the application program.

11. Conclusions and Future Work

In this paper, we have identified a set of desirable development support and required runtime support for SAW in trustworthy ubicomp application software, and presented an approach to providing this set of support with a dynamic reconfigurable SA middleware. The above support can greatly simplify the development and facilitate the execution and reconfiguration of trustworthy ubicomp application software with SAW and security requirements. We have implemented and evaluated our approach using RCSM. Our results are complimentary to OMG's work on middleware security, such as CORBA security services [17]. OMG's security specifications defined security protocols, mechanisms and services that can be utilized by business applications, whereas our approach provides an effective method with supporting tools to facilitate the specification and enforcement of SA security requirements of trustworthy ubicomp application software.

Since many malicious attacks, especially the attacks from inside users, on computing systems are conducted by abusing privileges granted to users or application software, one way to reduce malicious attacks is to enforce the "Least Privilege" principle. Our approach enables the specification and enforcement of SA security policies, in which the trust relationship among various entities and the privileges of various subjects in trustworthy ubicomp application software are dynamically changing under various situations. Hence, our approach facilitates the enforcement of the "Least Privilege" principle. In the future we plan to investigate how to incorporate SAW into existing defense techniques against denial-of-service and other attacks at the communication level, such as wireless communication injections and attacks on wireless network routing.

In addition, we plan to improve the expressive power of SA-IDL, develop a model-based approach for verification of SA-IDL specifications and a model-based test approach for cost-effective testing of the SAW aspect of trustworthy ubicomp application software, investigate effective dynamic scheduling techniques for scheduling situation-triggered actions, as well as the development and runtime support for security.

Acknowledgment

This work was supported in part by National Science Foundation under grant numbers ANI 0123980 and ITR-CYBERTRUST 0430565.

References

[1] S. S. Yau, Y. Wang and F. Karim, “Development of Situation-Aware Application Software for Ubiquitous Computing Environments”, *Proc. 26th IEEE Int'l Computer Software and Applications Conf. (COMPSAC 2002)*, August, 2002, pp. 233-238.

[2] S. S. Yau, et al., “A Smart Classroom for Enhancing Collaborative Learning Using Pervasive Computing Technology”, *Proc. 2nd ASEE Int’l Colloquium on Eng. Education (ASEE2003)*, 2003.

[3] S. S. Yau and F. Karim, “An Energy-efficient Object Discovery Protocol for Context-Sensitive Middleware for Ubiquitous Computing”, *IEEE Trans. on Parallel and Distributed Systems*, vol. 14(11), Nov. 2003, pp. 1074-1084.

[4] B. J. Nelson, “Context-Aware and Location Systems”, PhD thesis, University of Cambridge, Jan. 1998 http://www.sigmobile.org/phd/1998/theses/nelson.pdf

[5] A. K. Dey and G. D. Abowd “A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications,” *Human-Computer Interaction*, vol. 16(2-4), 2001, pp. 97-166.

[6] D. Caswell and P. Debaty, “Creating Web representations for places,” *Proc. 2nd Int’l Symp. on Handheld and Ubiquitous Computing (HUC2K)*, 2000, pp. 114-126.

[7] A. T .S. Chan and S. N. Chuang, "MobiPADS: A Reflective Middleware for Context-Aware Computing", In *IEEE Transactions on Software Engineering*, vol. 29(12), Dec 2003, pp. 1072-1085.

[8] C. Hess, M. Roman, and R. H. Campbell, “Building Applications for Ubiquitous Computing Environments,” Proc. Int’l Conf. Pervasive Computing, 2002; http://choices.cs.uiuc.edu/gaia.

[9] T. J. Lehman et al., “Hitting the Distributed Computing Sweet Spot with TSpaces,” Computer Networks, vol. 35(4), Mar. 2001, pp. 457–472.

[10] D. Bell and L. Lapadula, "Secure Computer System: Unified Exposition and Multics Interpretation", *Technical Report MTR-1997*, MITRE, Bedford, MA, 1975, pp. 134.

[11] B. W. Lampson, "Protection," *Proc. 5th Symp. on Information Sciences and Systems*, 1971, pp. 437-443.

[12] R. S. Sandhu, et al., "Role-based access control models," *Computer*, vol. 29(2), 1996, pp. 38-47.

[13] J. B. D. Joshi, E. Bertino, U. Latif and A. Ghafoor, "A generalized temporal role-based access control model," *IEEE Trans. on Knowledge and Data Engineering*, vol. 17(1), 2005, pp. 4-23.

[14] M. J. Covington, et al., "A context-aware security architecture for emerging applications," *Proc. 18th Annual Int'l Computer Security Applications Conf.*, 2002, pp. 249-258.

[15] D. G. Cholewka, R. A. Botha and J. H. P. Eloff, "A Context-sensitive Access Control Model and Prototype Implementation," *Proc. IFIP TX 11 16th Annual Working Conf. on Information Security*, 2000, pp. 53-66.

[16] S. S. Yau, Y. Yao and V. Banga, “Situation-Aware Access Control for Service-Oriented Autonomous Decentralized Systems,” *Proc. 7th Int’l Symposium on Autonomous Decentralized Systems*, 2005, pp. 17-24.

[17] OMG CORBA Security Service Specification v1.8, March 2002.

[18] C. Phillips, et al., "Security Engineering for Roles and Resources in a Distributed Environment," *Proc. 3rd Annual Information System Security Engineering Conf.*, 2002.

[19] S. S. Yau, D. Chandrasekar and D. Huang, "An adaptive, Lightweight and Energy-Efficient Context Discovery Protocol for Ubiquitous Computing Environments," *Proc. 10th Int'l Workshop on Future Trends of Distributed Computing Systems (FTDCS'04)*, May 26-28, 2004, pp. 261-267.

[20] S. S. Yau and F. Karim, "An Adaptive Middleware for Context-Sensitive Communications for Real-Time Applications in Ubiquitous Computing Environments", *Real-Time Systems*, vol. 26(1), 2004, pp. 29-61.

[21] S. S. Yau and F. Karim, "A Context-Sensitive Middleware-based Approach to Dynamically Integrating Mobile Devices into Computational Infrastructures", *J. Parallel and Distributed Computing*, vol. 64(2), Feb. 2004, pp. 301-317.

[22] R. Schantz and D. Schmidt, "Research Advances in Middleware for Distributed Systems: State of the Art", *Proc. IFIP 17th World Computer Congress*, August 2002, pp. 1-36.

[23] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. ACM*, vol. 20(1), 1973, pp. 46-61.

[24] C. Lu, et al, "Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms," *Real-Time Systems*, vol. 23(1-2), 2002, pp. 85-126.

[25] "Jini™ Technology Core Platform Specification", version 2.0, Sun Microsystems Inc., June 2003.
Available at: http://www.sun.com/software/jini/specs/core2_0.pdf

[26] "Naming Service Specification", version 1.3, Object Management Group, Inc., October 2004.
Available at: http://www.omg.org/docs/formal/04-10-03.pdf

[27] "Trading Service Specification", version 1.0, Object Management Group, Inc., May 2000.
Available at: http://www.omg.org/docs/formal/00-06-27.pdf

[28] S. S. Yau, et al., "Reconfigurable Context-Sensitive Middleware for Pervasive Computing," *IEEE Pervasive Computing*, 1(3), July-Sept. 2002, pp.33-40.