# Development of Situation-Aware Application Software for Ubiquitous Computing Environments

Stephen S. Yau, Yu Wang, and Fariaz Karim
Computer Science and Engineering Department
Arizona State University
Tempe, AZ 85287, USA
{yau, wangyu, karim}@asu.edu

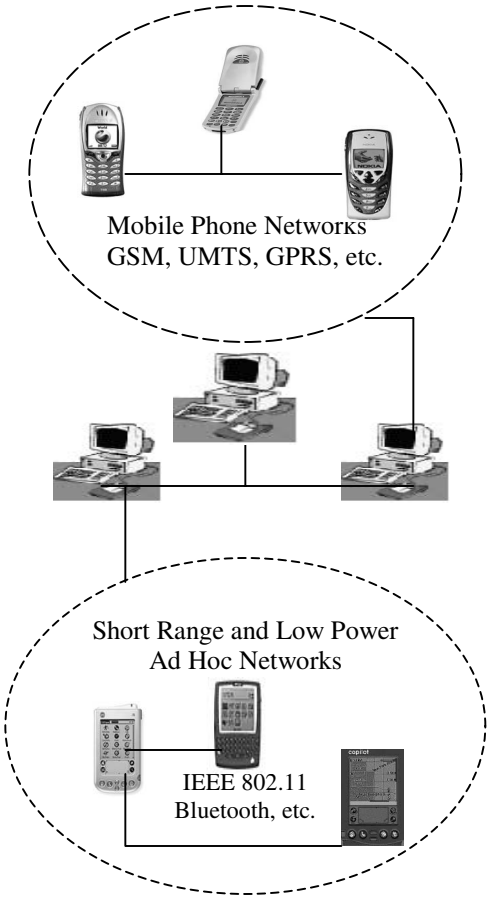
**Abstract**

Ubiquitous computing represents the concept of *computing everywhere*, making computing and communication essentially transparent to the users. Applications in this type of environments are context-sensitive. They use various contexts to adaptively communicate with each other across multiple network environments, such as mobile ad hoc networks, Internet, and mobile phone networks. The property of context-sensitivity often becomes inadequate in these applications, where combinations of multiple contexts and users' actions need to be analyzed over a period of time. Situation-awareness in application software is considered as a desirable property to overcome this limitation. In addition to being context-sensitive, situation-aware applications can respond to both current and historical relationships of specific contexts and device-actions. Currently, no well-defined concept of situation and no general method exist to facilitate the development of situation-aware application software for ubiquitous computing environments. In this paper, the concept of situation is formalized, and an approach to developing situation-aware application software is presented. The approach utilizes our Reconfigurable Context-Sensitive Middleware, and is illustrated by an example on *Smart Classroom*.

Keywords: Ubiquitous computing environments, situation-awareness, situation-aware interface definition language, reconfigurable context-sensitive middleware, mobile ad hoc networks, and *Smart Classroom*.

## 1. Introduction

Ubiquitous computing (ubicomp) [1] represents the concept of *computing everywhere*, making computing and communication essentially transparent to the users. Applications in this type of environments are context-sensitive, which means they use various contexts to adaptively communicate with each other in mobile ad hoc networks. Construction of this type of environments is now possible due to rapid progress in inexpensive, short range, and low-power wireless communication hardware and their continuing steady integration with various multi-protocol and multi-network environments. User applications in a typical ubicomp environment, as shown in Figure 1, have the following characteristics [2,3]:



❖ Context-sensitivity: Context-sensitivity is the capability of a device to detect its current context and changes in contextual data.
❖ Situation-awareness: Situation-awareness is the capability of a device to capture and analyze the relationship among multiple contexts and actions over a period of time.
❖ Ad hoc communication: Communication channels among application software tend to be instantaneously established and terminated due to changing contexts, device mobility, and resource availability.

While context-sensitive application software is adaptive, having only context-sensitivity is often inadequate if the application requirements need to consider the relationship between various contexts and history of users' and applications' actions over a period of time. For example, a location-aware tour guide as described in [4] is a context-sensitive application. It can be made more adaptive and intelligent if it can track and analyze individual user's past actions in different locations. Using such relationship between the context and actions enables application software to learn new information and use it with historical data to provide appropriate responses according to individual users' preferences. Clearly, situation-awareness implies context-sensitivity, but not vice versa.

Although the idea of situation has been discussed in literatures [5-7], they mostly end up as ad hoc and inconsistent treatments. In [5], situation is used to indicate user tasks, [6] refers situation as only an environmental context, and [7] considers a combination of these two. *Context Toolkit* [8] provides *context widget* components to isolate the details of context sensing from the application software. While this approach eases the effort required to develop context-sensitive applications, it does not provide a built-in capability to analyze multiple contexts and actions during execution. In [9] we presented a method to generate object-specific *context-reflectors*, using our Reconfigurable Context-Sensitive Middleware (RCSM) [10,11], capable of finding occurrence patterns of multiple contexts based on object interface specification. We also studied the effect of runtime change in context on objects' invocation and presented an algorithm to show how to dynamically adapt the sensitivity of the entire application without incurring any modification in individual objects.

In this paper we will define situation and present a method to develop situation-aware application software capable of analyzing both contexts and user actions, and to incorporate a mechanism in RCSM to enable objects to utilize their own situations in order to communicate with other situation aware objects.

## 2. Our Approach to Developing Situation-Aware Application Software using RCSM

Our approach to developing situation-aware application software for ubicomp environment includes the following major steps:

1. *To generate a situation-aware object interface specification.* To facilitate this, we have developed a Situation-Aware Interface Definition Language (SA-IDL) based on our definition of situations.
2. *To generate a Situation-Aware Adaptive Object Container (SA-ADC)* for runtime analysis and detection of application-specific situations.
3. *To generate the code of the situation-aware objects* whose interfaces are defined in Step 1.

The development service provided in our current Reconfigurable Context-Sensitive Middleware (RCSM) [10,11] is not sufficient to perform the above steps. In addition, during application software execution, we need to provide runtime services to i) periodically propagate the necessary sensor data to situation-aware application software, ii) discover compatible application objects in other devices, and iii) to provide a protocol for establishing remote communication links among situation-aware objects. Our RCSM architecture has the capability to perform i) and ii). We will extend RCSM architecture to address the above development and runtime services, and will refer this extended version as RCSM+.

We will elaborate Steps 1and 2 in Section 3, discuss the deployment and runtime services in RCSM+ in Section 4. We will include a brief discussion on our *Smart Classroom* test bed in each section to illustrate the corresponding steps.

## 3. Situation-Aware Interface Specification and SA-ADC Generation

Situation-aware interface reflects the application-specific situation-awareness requirements, that is, what situations to detect and what actions to take to respond to those situations. To specify an application-specific situation-aware interface in a file, we need to develop an SA-IDL. In this section, we will first present how we define situation and represent it formally, and then discuss how we specify an application-specific situation-aware interface and incorporate situation-awareness in application software. We will present how we generate the SA-ADC, its architecture and runtime detection of application-specific situation.

### 3.1 Situation Definition and Expression

Since situations use contexts, we introduce our definition of context first.

**Context:** We define *context* as any detectable and relevant attribute of a device, its interaction with other devices and/or its surrounding environment at an instant of time [10].

**Situation**: We define *situation* as an expression on previous device-action record over a period of time and/or the variation of a set of contexts relevant to the application software on the device over a period of time. Situation is used to trigger further device actions.

We will use an example to illustrate our approach step by step. This example is taken from our ongoing work on *Smart Classroom*, which is a ubicomp test bed to evaluate our approach in a real setting and to facilitate collaborative learning among college students. The following are two common features of a *Smart Classroom*:

- An instructor and his students use their own PDAs to collaborate by dynamically forming short-range mobile ad hoc networks in a classroom.
- Students' PDAs dynamically form mobile ad hoc networks based on proximity and specific contexts, such as location, light intensity, to collaboratively solve a specific problem.

Consider the following scenario of a *Smart Classroom*: Students in the class form small groups using their PDAs in the *Smart Classroom* to solve a specific problem. During group discussion, the instructor moves from one group to another to check the progress of each group. When the instructor walks towards a group (say, group K) and the walking has continued for 5 seconds, his PDA detects that he is interested in group K's discussion. This causes instructor's PDA to download the discussion material of group K.

In this scenario, the *situation* is "the instructor is moving to group K for 5 seconds (contexts variation) and his PDA has not downloaded the discussion material of group K (device-action record)".

It follows from our definition of situation that:

- Situation involves both contexts and actions.
- Some contexts may not be *directly* used for situation since some value used in situation cannot be *directly* collected through sensors or device, such as whether the instructor is moving to a specific location.
- Situation is only meaningful in a time range related to current time.

A situation expression system contains the following components:

- **Context Tuple:** We represent *context* as a tuple $<t, c_1, c_2, \ldots, c_n>$ [11], where $t$ is the time stamp and $c_1, c_2, \ldots, c_n$ are values of the context attributes relevant to the application software and collected through sensors, device and application software *directly*, such as light intensity, noise level, system time, application invocation rates, etc.

For the above example, we need to know the location of the instructor's PDA to analyze if the instructor is moving to group K. Hence, we define the context tuple as $<t, loc\_x, loc\_y>$, where (loc_x, loc_y) is a pair of coordinate values of the PDA location.

- **Action Tuple:** *An action tuple* has the same format of a context tuple and represents a device-action $<t, a_1, a_2, \ldots, a_n>$, where $t$ is the time when the action is taken and $a_1, a_2, \ldots, a_n$ are a set of attributes of the action, such as the action name, action parameters, etc.

For the above example, we need to check whether the PDA has downloaded the discussion material, so we define the action tuple as $<t, action\_name>$.

- **Derived Context:** Context tuples and action tuples are discrete samples of raw context and action data. In other words, they are the *direct* records of the environment and device information. In addition, we define *derived context* as a mathematical function of contexts that discovers how contexts vary with time. Note that the function could be recursive, which means a new derived context can be defined on existing derived contexts.

To check the situation described in the above example, we need to compute the distance between the instructor and the group K (assume the location of the group is constant $(x_{gk}, y_{gk})$), which is noted as *distance_i-gk*, and how the distance changes, which is noted as *distance_i-gk_change*. Hence, we have two derived contexts:

$distance\_i\text{-}gk = ((loc\_x - x_{gk})^2 + (loc\_y - y_{gk})^2)^{0.5}$

$distance\_i\text{-}gk\_change = \Delta\, distance\_i\text{-}gk$

- **Situation Expression:** A *situation expression* within a time range related to current time indicates how the contexts (may include both direct and derived contexts) and action vary. It has the following format:

$[\forall, \exists]$ t in <time range> ([context, action] <compare><value>)$^+$ (1)

Existing situations can form new situations by performing "not", "and" or "or" operations on them.

For the above example, the situation expression is:
*∀ t in <5 seconds prior to current time, current time> (distance_i-gk _change <0, action_name <> "download")*,
where *i* denotes the instructor's location and *gk* the location of group K.

### 3.2 Situation-Aware Object Interface Specification

We have developed a situation-aware interface definition language (SA-IDL). For the sake of brevity, we only list the following important SA-IDL grammar rules for our discussion:

1) <context_tuple> ::= 'ContextTuple' <ID> '{' 'time' ',' <variables> '}'
2) <action_tuple> ::= 'ActionTuple' <ID> '{' 'time' ',' <variables> '}'
3) <derived> ::= 'Derived' <ID> '{' <add_expr> '}'
4) <derived> ::= 'Derived' <ID> '{' <function> <ID> [<time-interval>] '}'
5) <situation> ::= 'Situation' <ID>'{' <quantifier> '('<time_range> ')' <comparisons > '}'
6) <situation> ::= 'Situation' <ID> '{' 'not' <ID >'}' | 'Situation' <ID> '{' <ID>'and' <ID> '}' | 'Situation' <ID> '{' <ID> 'or' <ID> '}'
7) <activate_tag> ::= '[' 'activate at' 'Situation' <ID > ']'

The following process is to specify the situation-aware interface in an SA-IDL file:

a. Find all the contexts and action attributes that will be used in the situations and put them in a context tuple and an action tuple respectively using Rules 1) and 2).
b. If the situation cannot use some context or action values directly, define a derived context for this purpose using Rules 3) or 4). In 3), <add_expr> is a mathematical expression of context or action values. In 4) the <function> is a pre-defined function of those values such as "delta", "sum", etc.
c. Use Rule 5) to express the situations related to a single time range. If needed, use Rule 6) to express more complicated situations by applying "and", "or" or "not" operations on defined ones.
d. Use Rule 7) to associate the situations and corresponding actions.

Figure 2 shows the situation-aware object interface specification in an SA-IDL file for the example in Section 3.1.

### 3.3 Application-Specific SA-ADC

```
interface collaborator {
    ContextTuple context_device1 {
        t,
        loc_x: integer;
        loc_y: integer;          }

    ActionTuple action_device1 {
        t,
        action_name: string;     }

    Derived distance_i-gk {
        ( (loc_x – x_gk)^2 + (loc_y- y_gk)^2)^0.5  }

    Derived distance_i-gk _change {
        Delta distance_i-gk    }

    Situation distance_i-gk _decreasing {
        ForAny (t > T-5)
        distance_i-gk_change<0
        action_name<> "download1"
        }

    [incoming] [activate at
        distance_i-gk_decreasing ]
        void download1 (string discussion)
}
```

Figure 2: The Situation-Aware Object Interface specification in an SA-IDL file for the instructor's PDA

When the SA-IDL file is compiled using the SA-IDL compiler, an SA-ADC is automatically generated. To recognize the situations defined in the SA-IDL file and activate the associate actions, an SA-ADC is composed of three modules: Base Class, Situation Analyzer and Dispatcher. The architecture is depicted in Figure 3.

**Base class** is the skeleton of the interface specified by the SA-IDL file. It contains method specification and some user-defined types, but no information of

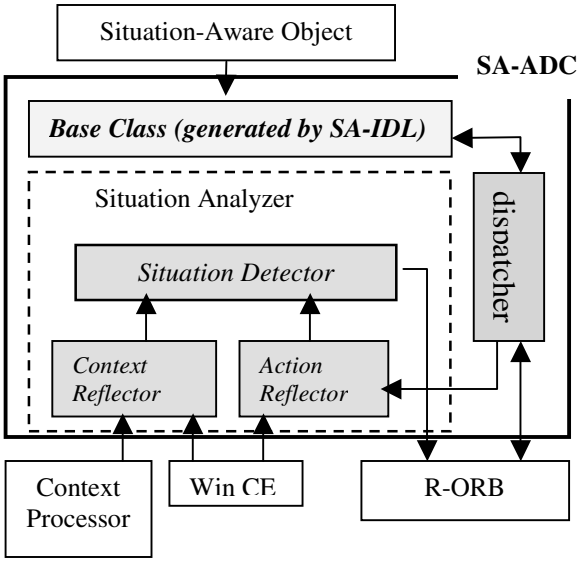



Figure 3: SA-ADC Architecture.

the situations defined in the SA-IDL file. The developer is responsible to generate the code of the situation-aware object that extends this base class.

**Situation Analyzer** is a module that analyzes contexts and actions and recognizes situations. It contains three components: Context Reflector, Action Reflector and Situation Detector. The reflectors get raw context and action data from the hardware context processor and operating system, and then form context tuples and action tuples as specified in the SA-IDL file and output the tuples to situation detector. The situation detector analyzes the data and recognizes the situations specified in the SA-IDL file. Once some situation is matched, situation detector will notify the R-ORB of the matched situation. This notification is called a *situation match event*. Then the R-ORB will start to check if a communication partner is available to trigger the associate action. If there is one, R-ORB will request the dispatcher to invoke that action.

**Dispatcher** is the module that invokes the actions that associated with situations. When it receives the activation request from the R-ORB, it defers the request to the base class. After the action is performed, the dispatcher sends the return values of the action to R-ORB.

For the SA-IDL file presented in Figure 2, the compilation generates an SA-ADC that is specific to it. The context reflector maintains a table of context tuples (*time, loc_x, loc_y*). The action reflector holds a table of action tuples (*time, action_name*). The situation analyzer computes and records the derived contexts (*distance_i-gk*, *distance_i-gk_change*) and measures the situation (*distance_i-gk_decreasing*).

## 4. Deployment and Runtime Services in RCSM+ to Support Situation-Aware Objects

In this section, we will discuss the deployment and runtime services in RCSM+ that enable situation-aware application objects to communicate with objects of other devices in an ad hoc network.

**Registration of Situation-Aware Objects:** The registration process is performed to deploy the application objects on top of our RCSM+ ORB (R-ORB). In addition to performing situation analysis on behalf of a situation-aware object, an SA-ADC acts as an adapter between the object and R-ORB to facilitate bi-directional communication. During the registration process every SA-ADC performs an initialization to bind it to a well-known interface of the R-ORB. The registration data includes situation-aware interface information of the application object, the sensor data required by the SA-ADC, and an upcall interface descriptor of the SA-ADC.

**Sensor-Data Propagation to Situation-Aware Objects:** During application software execution, an SA-ADC needs to have timely access to the latest sensor data to perform object-specific situation analysis. The *context-processor* component of R-ORB performs periodic data acquisition from the sensors connected to the device. Using the data collected during registration process, R-ORB categorizes and propagates the sensor data to appropriate SA-ADCs using the corresponding upcall interfaces. This process uniformly isolates the SA-ADCs from the low-level data acquisition routines connected to the sensors. In our *Smart Classroom* test bed, four different types of sensor data are collected: noise, light, motion, and location.

**Situation-Aware Communication Channel Establishment:** A situation-aware communication channel (SCC) between two devices is a type of channel that is established and terminated based on application-specific situation specification. During application software execution, R-ORB performs proactive device discovery and uses a protocol to establish and maintain an SCC with a remote device. The protocol establishes a new channel based on the following conditions:

- **Reachability:** There exist two devices A and B such that they are in the transmission range of each other.
- **Existence of Situation-Aware Objects:** Each device (i.e. A and B) has at least one situation-aware object.
- **Recent Occurrence of Situation-Match Events:** There exists at least one SA-ADC component in both A and B that has recently generated a situation-match event. This implies that the application object associated with such an SA-ADC has at least one method that can be invoked currently based on its situation-specification. Let these objects be $O_a$ and $O_b$ and their methods be $M_a$ and $M_b$ for Devices *A* and *B* respectively.
- **Compatibility:** $M_a$ and $M_b$ are suitable to exchange data with each other in the sense of matching of interface signatures, including the number and types of parameters, or other application-specific criteria, such as compatibility of radio-frequency identifiers, security attributes, etc.

The protocol is symmetric and uses *peer-to-peer* interaction semantics. Any device in a network can independently check the validity of the conditions and establish a communication channel. This strategy promotes spontaneous and ad hoc networking among the devices. Now we would like to explain how situation-aware communication channels are established in our *Smart Classroom* scenario presented in Section 3:

i) During runtime when appropriate situations are valid, the SA-ADCs in both PDAs generate *situation-match* events and notify the respective R-ORB.

ii) Each R-ORB then uses the information from Step i) to initiate the service discovery procedure. If other PDAs are in the transmission range, instructor's PDA initiates service discovery to discover students' PDAs, and vice versa.

iii) If the discovery is successful, the respective R-ORBs generate service-match events. The service-match event is used to trigger the initiation of a new situation-aware communication channel between R-ORBs of both of PDAs.

iv) After the communication channel is established, each R-ORB requests the respective *dispatcher* of the instructor and student-group application software to activate the corresponding objects. Following the invocation of student-group object, the data is passed through the dispatcher object.

## 6. Discussion and Future Work

In this paper, we have presented an approach to situation-aware application software development for ubicomp environments. Specifically, we have presented how our SA-IDL is used to define situation-aware object interfaces and to generate application-specific SA-ADCs. We have also discussed how the object request broker of RCSM+ (R-ORB) facilitates situation-aware communications among devices. We are currently incorporating our results into our *smart classroom* test bed to evaluate RCSM+. The SA-IDL compiler and SA-ADC components are being developed in C++ for Windows CE-based PDAs. R-ORB is being co-designed in software and Xilinx Spartan II reconfigurable hardware. For more detailed information, refer to our project web site: http://www.eas.asu.edu/~rcsm.

Future research in this area includes using the situation-awareness capability in RCSM+ to develop group communication and information dissemination services for ubicomp environments. Effective testing of situation-aware application software developed using our approach is also needed. In addition, we will extend SA-ADCs to provide priority-based object activation for situation-aware real-time software.

## Acknowledgement

This research is supported by National Science Foundation (NSF) under grant number ANI-0123980.

## References

[1] M. Weiser, "Some Computer Science Problems in Ubiquitous Computing", *Communications of the ACM,* Vol. 36, No. 7, July 1993, pp. 75-84.

[2] G. Abowd and E. D. Mynatt, "Charting Past, Present, and Future Research in Ubiquitous Computing", *ACM Trans. Computer Human Interaction*, Vol. 7, No.1, pp. 29-58, March 2000.

[3] The Oxygen Project, http://www.oxygen.lcs.mit.edu/.

[4] G. Abowd, et al, "Cyberguide: A Mobile Context-Aware Tour Guide", *Wireless Networks*, Vol. 3. No. 20, pp. 421-433, 1997.

[5] P Marti, F. Gabrielli, L. Petroni, and F. Pucci, "Situated Interactions in Art Settings", *Proc. Workshop on Situated Interaction in Ubiquitous Computing at CHI2000*, April 2000, http://www.teco.edu/chi2000ws/papers/29_marti.pdf

[6] B. Schiele, T. Starner, B. Rhodes, B. Clarkson and A. Pentland, "Situation Aware Computing with Wearable Computers." *Augmented Reality and Wearable Computers*, W. Barfield and T. Caudell (ed.), Lawrence Erlbaum Press. 1999.

[7] T. Selker and W. Burleson, "Context-Aware Design and Interaction in Computer Systems", *IBM System Jour.*, vol. 39, no. 3-4, 2000, http://cac.media.mit.edu:8080/contextweb/jsp/index.htm.

[8] A. K. Dey and G. Abowd, "The Context-Toolkit: Aiding the Development of Context-Aware Applications", Workshop on Software Engineering for Wearable and Pervasive Computing, Ireland, 2000, http://www.cc.gatech.edu/fce/contexttoolkit/.

[9] S. S. Yau and F. Karim, "Context-Sensitive Software Development for Ubiquitous Computing Environments", *Proc. 25th Int'l Computer Software and Applications Conference (COMPSAC 2001)*, USA, October 2001, pp. 263-268.

[10] S. S. Yau and F. Karim, "Reconfigurable Context-Sensitive Middleware for ADS Applications in Mobile Ad Hoc Network Environments", *Proc. 5th IEEE Int'l Symp. Autonomous Decentralized Systems (ISADS 2001)*, March 2001, USA, pp. 319-326.

[11] S. S. Yau and F. Karim, "Context-Sensitive Middleware for Real-Time Software in Ubiquitous Computing Environments", *Proc. 4th IEEE Int'l Symp. Object-Oriented Real-Time Distributed Computing (ISORC 2001)*, May 2001, Germany, pp. 163-170.