Context-Sensitive Distributed Software Development for Ubiquitous Computing Environments

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

Abstract

Ubiquitous computing represents the next wave of distributed computing, which focus on integrating computers with various wearable, mobile, and sensor devices, thus making computing and communication essentially transparent to the users. This type of environments mainly consists of myriad of embedded computing nodes interacting in transparent fashion to provide different services to the users. In addition to being resource-constrained, applications in these environments are context-sensitive and often operate in mobile ad hoc networks. Although different types of context-sensitive applications with diverse focus have been developed for ubiquitous computing environments, no general methods are available for developing application software for the above environments. In this paper, the development of context-sensitive distributed software for ubiquitous computing environments is presented.

Keywords: Ubiquitous computing environments, context-aware software, context-sensitivity, reconfigurable context-sensitive middleware, mobile ad hoc networks

1. Introduction

The trend in distributed computing is moving toward ubiquitous computing (ubicomp) environments [1], where a vast number of embedded, wearable, and handheld devices communicate transparently to provide different services to the users. These devices mostly use low power and short-range wireless communication capabilities. By utilizing multiple on-board sensors and other information about the surrounding environment, these devices often establish context-triggered communication links [2]. As mentioned in [3, 4], applications that execute in ubiquitous computing environments should have the following properties:

- Context sensitivity: Applications use various data about the surrounding environment and the device to adapt their behavior.
- Ad hoc communication: Due to the mobility and increased transparency of the devices, applications mostly participate in ad hoc communication with other devices to exchange information.

The properties related to context-sensitivity and ad hoc communication are basic characteristics of context-sensitive distributed software. These properties require both effective system support and a general development method. In [2,3], we addressed the system aspect of these issues by presenting Reconfigurable Context-Sensitive Middleware (RCSM), which provides a Context-Aware Interface Definition Language (CA-IDL) for defining context-aware object interfaces. In addition, RCSM includes R-ORB, which is a software-hardware co-designed context-sensitive Object Request Broker to provide context-triggered object activation. From the application development perspective, however, little work has been done to address a software development methodology for applications in ubicomp environments. Majority of the existing work, such as location-aware tour guides, smart wearables, etc. [5-10], are mostly focused on developing individual ubicomp applications, rather than providing a general development framework. In this paper, we will present an approach to developing context-sensitive distributed software for ubicomp environments. This approach utilizes our ongoing work on RCSM [2,3], which provides the underlying system support.

2. Our Approach

Our approach assumes that the necessary objects and classes are already identified from the requirements using any of the existing methods on object-oriented analysis and design. Based on the nature of the context-awareness of the target application, our approach associates the desired context-awareness with the necessary objects. It also provides a scheme to derive the initial context-sensitivity and to adapt the degree of context-sensitivity of an application during runtime. This approach can be summarized in Figure 1 with the following steps:

0-7695-1372-7/01 $10.00 © 2001 IEEE

1) Context-awareness is associated with individual object methods. As such, application objects are activated based on contexts, in addition to explicit user invocation. This type of activation is referred to as context-triggered object activation.
2) A *context-reflector* (CR) for each context-aware object is generated based on the procedure in Step 1). A *CR* performs the necessary processing of contextual data on behalf of the application object.
3) The initial context-sensitivity of each context-aware object is derived to specify the initial behavior of the applications.
4) The context-sensitivity is continuously refined during runtime to address changing contextual conditions during operations of the device. The new context-sensitivity of an object is recomputed based on runtime data. To support the context-triggered communication of the objects with the remote

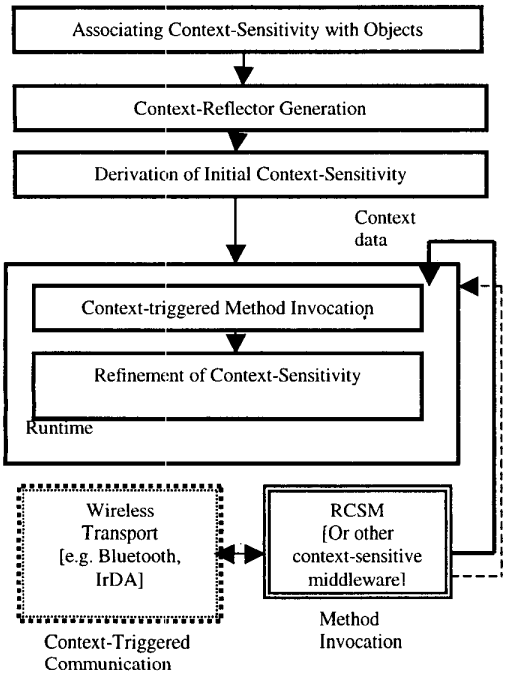

Figure 1: Our approach to development of ubicomp applications software.

devices, RCSM [2,3] is used. RCSM uses wireless transport protocols, such as Bluetooth, IrDA, etc.

In the remainder of the paper, we will elaborate each of these steps in Sections 3-6, and provide an illustrative example.

3. Associating Context-Sensitivity with an Object

In this section, we will discuss how to make the objects of an application software context-aware:

a) The desired context [2,3] of the application software is determined from the system requirements. Objects, which are only directly invoked by the underlying middleware during connection establishment [3], need to be associated with context-sensitivity.
b) For each method of a context-aware object from Step a), the desirable context is then specified as a context-tuple using the *RCSMContext_var* construct of our Context-Aware Interface Definition Language (CA-IDL) [3]. For complex specification involving multiple contexts, the *timed regular expressions* [3] are used.
c) The specified context is associated with a particular method of an object using two simple rules as specified in [3]. These rules are summarized here for the sake of completeness:

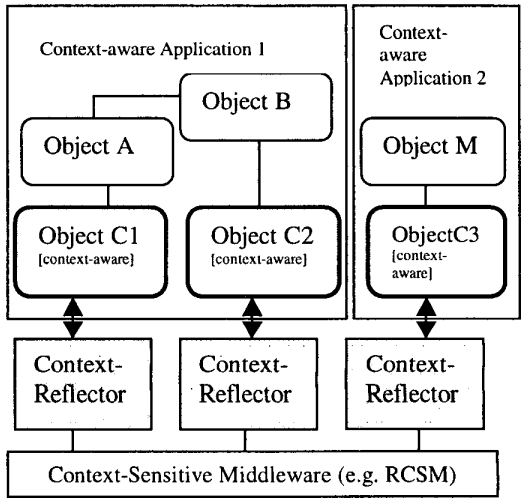

Figure 2: An architecture of context-aware distributed application software using our approach.

- Each method is annotated with either *[incoming]* or *[outgoing]* tags. An *[incoming]* tag signifies that the corresponding method must receive an external event or data to initiate its operation. An *[outgoing]* tag signifies that the method can be activated without any external event or data as long as the desired context remains true in the surrounding environment.
- Each method is also annotated with *[activate-at-context x]* tag, where x represents a context-tuple or *timed-regular expression*, as described in [3].

Figure 2 shows the structure of a context-aware application based on our approach. In this figure, Objects C1, C2, and C3 are context-aware according

to Step a) described previously. C1 and C2 are included in a single application while C3 is part of another application. All of these applications are supported by a context-sensitive middleware, such as RCSM [2,3]. A *context reflector* invokes each object during a connection establishment process. Each *context reflector* uses the context-data from the device to determine the suitability of object activation, which depend on both *context-match* and *identity-match* [2,3] events from RCSM.

4. Context-Reflector Generation

Context reflectors (CR) are object adapters that connect context-aware objects, as identified using the procedure in Section 3, with the underlying RCSM. Traditional object adapters used in existing middleware specifications, such as CORBA, invoke objects based on explicit requests from the clients. These object adapters are not capable of detecting the appropriate contexts based on the context specification of the object methods. CRs are thus necessary to address this issue. The following steps are used to generate a CR:

1) For each specified context in the object method, a timed finite state machine (timed-FSM) is generated. A timed-FSM is a deterministic FSM with a timer-state, which accepts *timer-init* and *timeout* events. Timer-FSM has two final states – *matched* and *not matched.*
2) To observe any temporal pattern in contexts, a timed regular expression is used to construct a timed-FSM for the following operators: +, ^, *, and -> [3]. The generated timed-FSM is used to detect if a desirable context is true at any instant of time. The use of timed-FSM inside the *CR* is referred to as *context-detection process.*
3) The frequency of *context-detection process* is set according to the intended frequency of the corresponding *CR.*

The remaining portion of our CR generation process involves associating a CR object with an ADC component [3].

5. Deriving Initial Context-Sensitivity

Initially, as the application software is being developed and deployed, no contextual data exists to facilitate the runtime operation of the application software. As such, it is necessary to initially assign context-sensitivity to the objects. The initial context-sensitivity, however, can be adapted at runtime in response to the changing environments. We will describe the adaptation process in the next section.

The derivation of initial context-sensitivity is divided into two major steps. First, we quantifiably define the notion of context-sensitivity. Second, we compute the initial context-sensitivity of an object using this definition.

Context-Sensitivity: Context-sensitivity represents both how fast a context-aware object reacts to its surrounding contexts and how it behaves in response to the variation in the context data. We established this concept in [3]. For the purpose of clarity, we refer to the j^{th} method of object q as δ^q_j. Furthermore, we define Δ^q as the set containing all the methods δ^q_j.

First, we define the context-sensitivity of a *context-reflector* as follows:

$$CS^q_{cr} = {CS_{rcsm}}/{\gamma^q_{cr} D^q_{cr}}, \quad (1)$$

where CS^q_{cr} represents context-sensitivity of the context-reflector *CR* of object q; CS_{rcsm} is the context-sensitivity of the underlying RCSM; γ^q_{cr} is the period of *context-detection* process; $D_{cr}{}^q$ is the time taken by CR to perform context-detection process. In case the period of detection process reaches ∞, CS^q_{CR} becomes 0. In (1), $D^q_{cr} \geq 1$ and $\gamma^q_{cr} \geq 1$. Now, using (1) we define the context-sensitivity of a method δ^q_j as follows:

$$CS^q_j = \frac{CS^q_{CR}}{D_j T_j N_c} \sum_{i=1}^{N_c} \left| \frac{C_i^{k+1} - C_i^k}{C_i^r - C_i^l} \right|, \quad (2)$$

where $CS^q{}_j$ is the context-sensitivity of δ^q_j; D_j is the delay between a context-match event and the invocation of δ^q_j; T_j is the time taken by δ^q_j to complete its operation; N_c is the number of unique contexts associated with δ^q_j; $C_i{}^r$- $C_i{}^l$ is the range of i^{th} context in which δ^q_j's behavior remains the same; $C_i^{k+1} - C_i^k$ is one unit of context value for the i^{th} context. In (2), $D_j \geq 1$ and $T_j \geq 1$.

Now, we can compute the context-sensitivity of the entire object CS^q as follows:

$$CS^q = \frac{1}{N_q} \sum_{j=1}^{N_q} CS^q_j, \quad (3)$$

where N_q is the total number of context-aware methods in q (i.e. $N_q = |\Delta^q|$).

According to (1), (2), and (3), context-sensitivity of an object q depends on the following factors:

- The context-sensitivity of all the methods in q.
- The delay between a method invocation and its corresponding *context-match* event.
- The time taken by a method to complete its task.
- The period of the context-detection process.
- The variation of the context-data in which a method can be invoked.
- The context-sensitivity of the underlying RCSM.

Initial Context-Sensitivity: The context-sensitivity of a method δ^q_j can be deduced statically if proper values of D_j and γ^q_{cr} are available. We ignore the derivation of CS_{rcsm} in this paper. Assuming a specific value of CS_{rcsm}, we use the following technique to determine the values of D_m and γ^q_{cr}:

A) Assume that each method j is assigned a priority value with respect to other methods in object q. Let this priority be C_j. Let C_{max} be the highest priority.
B) Upgrade the priority of j if the probability of *simultaneous context-match* events for both δ^q_j and a higher priority method δ^q_k is 0:

For each method j *in* q,

Let $S_j = \{\forall \delta^q_k \mid C_{\max} \geq C_k > C_j\}$;

Let $S^{\Phi}_j = \{\forall \delta^q_m \mid C_j > C_m\}$;

1. $C'_j = C_j$.
2. *Choose a* $\delta^q_w \in S_j \mid \forall \delta^q_i : \delta^q_i \neq \delta^q_w \wedge C_i > C_w$
3. *if* $P(\eta_w \mid \eta_j) = 0$
   $C'_j = C_w$.
   $\forall m : m \in S^{\Phi}_j, C_m = C_m + 1$.
4. $S_j = S_j - \{\delta^q_w\}$.
5. *Go to Step 2 if* $S_j \neq \{\phi\}$.
6. $C_j = C'_j$. *Stop*.

C) Calculate the relative weight of j W_j:

$$W_j = \left[1 - \left(C_j / C_{\max}\right) \prod_k P(\eta_k \mid \eta_j)\right] \quad (4)$$

where $P(\eta_k \mid \eta_j)$ is the probability of the occurrence of context-match events for δ^q_k when the current context matches the context of method δ^q_j. Here, k satisfies the condition: $\forall k : 1 \leq k \leq N_m \mid \delta^q_k \in \theta^q \wedge \langle C_j \geq C_k \rangle \wedge P(\eta_k \mid \eta_j) > 0$.

D) Derive γ^q_{cr} simply as follows:

$$\gamma^q_{cr} = \sum_i^{N_q} T_i, \quad (5)$$

where T_i is the time required to complete δ^q_i; N_q is already defined in (3).

E) Derive D_j as follows:

$$D_j = \sum_k T_k, \quad (6)$$

where W_k corresponds to the weight of δ^q_k from (4). Here, k is constrained by the condition: $\forall k : 1 \leq k \leq N_m \mid W_k \geq W_j \wedge \delta^q_K \neq \delta^q_J$.

In (4), the weight is decreased if the probability of invocation of higher priority methods is non-zero. This information is used in (6) to compute the delay of individual methods in q. Combining the context-sensitivity of a *context-reflector* and the RCSM with an object q decreases q's sensitivity. This is true in reality, since the overhead of additional layers between the application software and the sources of the contexts (e.g. sensors) in fact increases the time taken to react to a *context-match* event.

6. Runtime Change in Context-Sensitivity

Changing the context-sensitivity of an object during runtime is necessary to adapt the device in different environmental conditions. For example, by reducing the context-sensitivity of an object, a device can conserve resources in case of limited battery power, or increasing the context-sensitivity of a personal communication object may lead to fast response time in appropriate locations, such as a public meeting place. While it is possible to explicitly change the sensitivity by the user, in this section we focus on the automated adaptation in context-sensitivity by utilizing the tradeoff between the cost of delaying the invocation of a method and the cost of ignoring a *context-match* event completely.

The principal idea is to change the period (i.e. γ^q_{cr}) of *context detection* process. The following procedure is used for changing γ^q_{cr}, which essentially affects the context-sensitivity:

- Let L_n be the number of context-match events in the current cycle and L_p be the number of context-match events occurred in the previous cycle.
- If $L_n < L_p$, fewer methods need to be invoked. In this case, the *context-reflector* reduces the context-sensitivity by increasing γ_{cr}^{q} to $\gamma_{cr}^{q'}$:

$$\gamma_{cr}^{q'} = \frac{\gamma_{cr}^{q} L_n}{L_p} \tag{7}$$

- If $L_n > L_p$, more methods need to be invoked than previously. A decision needs to be made between the costs of not invoking some of the methods as oppose to delaying some of them. Let these costs be C_{ni} and C_d respectively. Then, the context-sensitivity is changed by changing γ_{cr}^{q} as follows:

$$\gamma_{cr}^{q'} = \frac{\gamma_{cr}^{q} L_p}{L_n} \quad if\ C_{ni} > C_d$$
$$= \gamma_{cr}^{q} \quad otherwise \tag{8}$$

7. An Illustrative Example

In this example, we illustrate the method presented in this paper by showing how a location-aware ubicomp application software can be developed for different devices. The requirements are as follows:

R1: The entire system is a network of public information services. Users are equipped with different devices to access different information. Two different types of devices A and B are used.

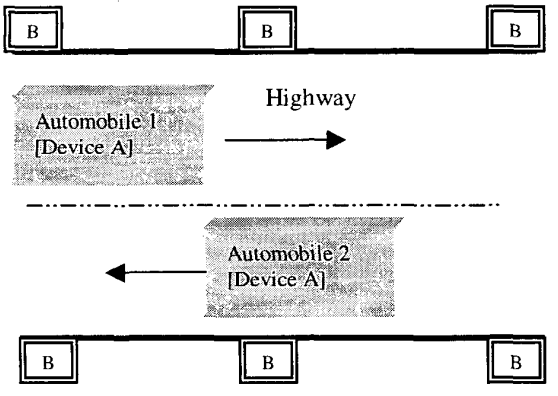

Figure 3: Devices A and B in a highway in the example.

R2: Type A devices are installed in automobiles. They access highway status data at different locations while the automobiles are moving.
R3: Type B devices are installed along the highways (every 0.2 mile) to broadcast the highway status data.

R4: Communication among the sensors occur based on the following conditions:

Devices A's exchange data with Devices B's whenever Devices B's are within the range. Due to the mobility of Devices A's, it is possible that multiple Devices B's can provide the same data to a single Device A. The scenario is shown in Figure 3.

Device A includes highway-status reporting software (implemented as Object A), which periodically displays different information about the current highway. Examples of data include traffic jam, alternative routes, accident information, etc. The object needs to be activated when the automobile is on a highway and a Device B is in 15 feet range. This information can be represented as a context using the location of the actual highway. Assume that each transfer of highway data from a Device B to a Device A requires 1.2 seconds.

Devices B's are installed in every 0.2 miles on both sides of the highway. These devices (implemented as Object B) communicate with the backend servers to gather real-time traffic data and broadcast them when Devices A's are in range.

Step 1) Associating Context-Sensitivity with Objects A and B: Objects A and B are the only objects used in Device A and Device B respectively. Their interfaces are defined in Context-Aware Interface Definition Language (CA-IDL) [3], which is a superset of CORBA 2.4.2 Interface Definition Language specification. CA-IDL uses additional language constructs to associate a method of an object with a particular context-tuple.

- Interface of Object A-Highway-Status-Receiver

```
//Name: Highway-Status Download Object
//ID: 001
//Define a context variable
RCSMContext_var DeviceContext
C_A1 where
Location == "Highway 101"and
DistanceAdjacentDevice <= 20.
//Method 1: activate when Device B is in range
[incoming] [activate at C_A1]
download_data([in] string data)
```

- Interface of Object B-Highway-Status-Distributor

```
//Name: Highway-Status Distributor Object
//ID: 002
//Define a context variable
RCSMContext_var DeviceContext
C_B1 where
```

```
DistanceAdjacentDevice <= 20.
//Method 1: activate when Device A is in range
[outgoing] [activate at C_B1]
distribute_data([out]stringdata)
```

Method1 in Device B only requires another device within 15 feet range to be activated. The actual type of highway is irrelevant here since its location is fixed along the highway as shown in Figure 3.

Step 2) Context Reflector Generation: Two independent *context reflectors* are generated based on the context-specifications of the interfaces. The first *context reflector* identifies if context C_A1 becomes true, while the second reflector checks if context C_B1 becomes true. The input to both context reflectors come from the underlying RCSM.

Step 3) Assigning Initial Context-Sensitivity: The period of *context-detection* process is initially set at 60 seconds for both devices. Using (6), the initial delay parameters of both methods are:

$D^{object\ A}_1$ = 6 seconds and $D^{object\ B}_1$ = 1second.

Here, we assume the C_j parameters of the methods are 5 and 3 respectively. In addition, the value of C_{max} is 5 in both devices, and the values of P (η_k | η_j)'s are 0.1 and 0.3 respectively.

Step 4) Runtime Change in Context-Sensitivity: Consider Device A's automobile moves along the highway at a speed of 60 miles/hour. Since a Device B is installed in every 0.2 miles, a *context-match* event will occur in every 12 seconds. As specified before, a single download takes 1.2 seconds and the period of context-detection process is 60 seconds. In this case, Device A decreases the frequency of context-detection process to approximately 12 seconds using the equation in (7).

6. Discussion

In this paper, we have presented an approach to development of object-oriented distributed software for ubiquitous computing environments. Specifically, we have presented techniques for associating context-sensitivity with object methods and deriving the initial context-sensitivity of individual objects. In addition, we have also discussed how the context-sensitivity can be easily changed at the *context reflector* level without affecting application objects. This approach utilizes a context-sensitive middleware, such as our RCSM, to receive the necessary system support. A benefit of our approach is that except the CA-IDL specification, application developers do not need to additional code to manage the context-sensitivity. This results in simpler design and rapid development time.

Currently, our research in this area is focused on refining the CA-IDL (part of our RCSM research) to allow definitions of the interfaces of situation-aware applications. In addition, we are also improving the technique to change the context-sensitivity during runtime by incorporating how fast the surrounding context changes over a period of time. In this regard, we are in the process of quantifying the dependency between context-sensitivity and the rate of context change. Future research will include evaluation and experimentation of the techniques using our RCSM prototype and middleware support for dynamically managing context-sensitivity.

References:

[1] M. Weiser, "Some Computer Science Issues in Ubiquitous Computing, *Comm ACM*, Vol. 36, No. 7, pp. 720-84, July 1993.

[2] S. S. Yau and F. Karim, "Reconfigurable Context-Sensitive Middleware for ADS Applications in Mobile Ad-Hoc Network Environments", *Proc. 20th IEEE Int'l Symp. Autonomous Decentralized Systems (ISADS 2001)*, pp. 319-326, March 2001.

[3] 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)*, pp. 163-170, May 2001.

[4] 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. 29-208, March 2000.

[5] R. Want, et al, "The PARCTab Ubiquitous Computing Experiment", *Xerox PARC Technical Report, CSL-95-1*, http://nano.xerox.com/parctab/csl9501/paper.html.

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

[7] R. Oppermann and M. Specht, "Adaptive Support for a Mobile Museum Guide", *Proc. Interactive Applications of Mobile Computing (IMB 98)*, 1998.

[8] N. Marmasse and C. Schmandt, "Location-Aware Information Delivery with comMotion", *Proc. 2nd Int'l Symp. Handheld and Ubiquitous Computing (HUC 2000)*, pp. 157-171, September 2000.

[9] P.R. Chesnais, "Canard: A Framework for Community Messaging", *Proc. 1st Int'l Symp. Wearable Computers (ISWC 97)*, http://www.computer.org/proceedings/iswc/8192/8192toc.htm.

[10] G. Chen and D. Kotz, "A Survey of Context-Aware Mobile Computing Research", Technical Report TR2000-381, Dartmouth College, Dept. of Computer Science, November 2000, http://www.cs.dartmouth.edu/~dfk/papers/#mobile.