Context-Sensitive Middleware for Real-time Software In 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

Context-awareness is increasingly becoming an important capability in devices for ubiquitous computing environments. These devices use on-board sensors and history of user interaction to collect data that are used to adapt their behavior to suit with the current environment. There is a need to support real-time software in ubiquitous computing environments, especially in reactive systems, such as distributed and mobile sensors, location-based information services, etc. In these cases, both behavior and the interaction among devices depend on constantly changing environmental conditions, in addition to explicit user control. This characteristic requires specific system services to support the development and the runtime operation of real-time context-aware software. This implies that the underlying services must themselves be context-sensitive. In this paper, Reconfigurable Context-Sensitive Middleware (RCSM) is presented to facilitate real-time context-aware software in ubiquitous computing environments.

Keywords: Context-sensitive middleware, ubiquitous computing, context-aware applications, context-triggered communication.

1. Introduction

Ubiquitous computing environments [1] focus on integrating computers with the physical environment, making computing and communication essentially transparent to the users. Devices that operate in these environments are usually embedded and use low-power and short-range wireless communication capabilities. In addition, the devices are free to move arbitrarily (e.g. a wearable device moves with a living carrier), usually have bandwidth and energy constraints, and are equipped with multiple sensors. The topologies in ad hoc networks are dynamic, and usually no dedicated network connectivity devices exist. Based on physical environmental conditions and other stimulus, nodes in this environment form numerous webs of ad hoc short-range wireless networks to exchange information, and react in a transparent fashion. As described in [2], distributed applications executing in these devices should have the following capabilities:

1. Context and resource sensitivity: Applications use various data about the surrounding environment to adapt their behavior and interactions.
2. Impromptu and volatile distributed interaction: Communication channels among applications tend to be instantaneously established and terminated due to changing contexts and node mobility.
3. High-density: As the number of mobile computing nodes increase, the volume of applications interaction tend to increase exponentially.

Based on the above characteristics, it is necessary to provide system support for the development and runtime support for applications that require real-time capabilities as well as context-awareness. However, this requirement implies that the underlying system services must themselves be context-sensitive (or context-aware). In this respect, middleware can be very effective to provide the support if they can reduce the effort required to develop distributed software and runtime services for applications with the abovementioned characteristics, in addition to providing the normal services, such as interoperability, location transparency, naming service, etc. However, the current state of the art in middleware is not adequate to support context-awareness for real-time software in ubiquitous computing environments. Current real-time middleware, such as TAO and dynamicTAO [3,4], are appropriate for real-time and resource-constrained systems, but are not suitable to work with the emerging networking technologies for ubiquitous computing environments, such as Bluetooth, IrDA, HomeRF, PicoNet, Personal Area Networks, and MANET routing protocols, such as AODV, DSDV, etc. [5-11]. In addition, these middleware implementations do not provide the facilities for context-aware software. A context-sensitive middleware was presented in [2,12] to support context-aware Autonomous Decentralized Systems

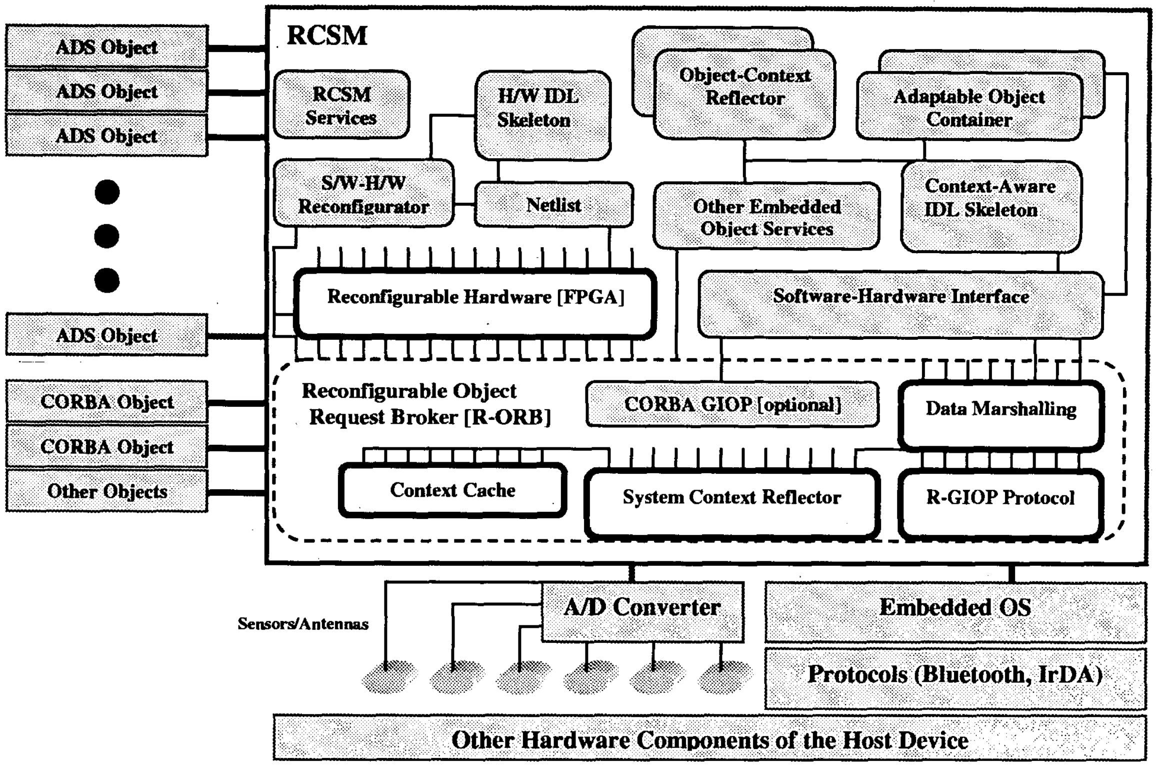

Figure 1: A High-Level Architecture of RCSM.

(ADS) in mobile ad hoc networks (MANET) [13]. However, it does not provide the necessary facilities for real-time software. On the other hand, ALICE [14] provides system services to allow CORBA objects running on mobile devices to interact transparently, but it also does not address context-awareness and real-time issues. In this paper, we will present an approach to support the development and runtime operation of context-sensitive real-time software using an object-oriented embedded middleware. This approach, which is compliant with the OMG CORBA specification, extends Reconfigurable Context-Sensitive Middleware (RCSM) [2,12] to provide context-sensitive services for real-time software.

2. Our Approach

Before we present our approach, it is necessary to provide an overview of RCSM [2,12]. RCSM is a context-sensitive middleware, which means it uses the contextual data of a device and its surrounding environment to initiate and manage ad hoc communication with other devices.

As shown in Figure 1, RCSM is co-designed in software and hardware, and combines the power of abstraction provided by CORBA with reconfigurable hardware to support ADS applications in MANET environments [2,12]. Specifically, RCSM is an embedded middleware with the following characteristics:

a. Uses dedicated reconfigurable Field Programmable Gate Arrays (FPGA) to provide core middleware services.
b. Provides a context-based reflection and adaptation triggering mechanism.
c. Provides an Object Request Broker and Inter-ORB protocol that are context-sensitive and invokes remote objects based on contextual and environmental factors, thereby facilitating autonomous exchange of information.
d. Can be used to supplement existing middleware specifications, such as CORBA and COM.

The design of RCSM is partially based on OMG CORBA and RT-CORBA specifications. The rationale behind taking a software-hardware co-design approach to designing RCSM for embedded devices is presented in [2]. This approach to developing RCSM can be compared with the micro-kernel approach to developing operating systems. In case of RCSM, the *micro-kernel* part is implemented in reconfigurable hardware, and it consists of the R-

ORB, *Device Context Reflector*, and data-marshalling units. Other components, including the equivalent of CORBA Portable Object Adapter (POA) or Component Models (CCM), are implemented in software to achieve flexibility to address a variety of functionality and capabilities of embedded devices.

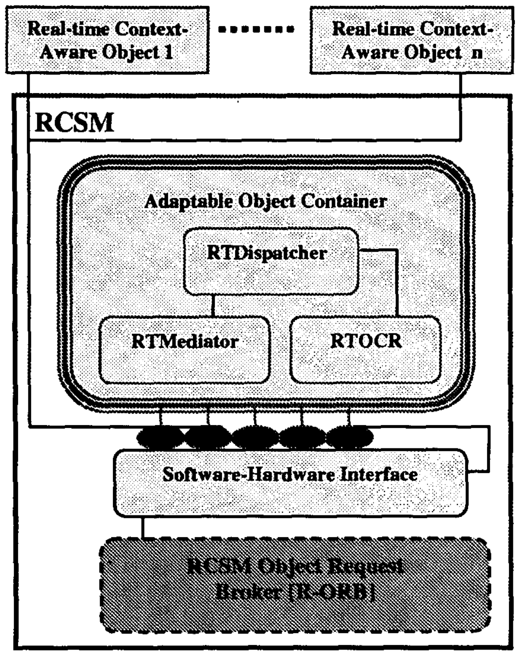
Real-time Context-Aware Object 1
Real-time Context-Aware Object n
RCSM
Adaptable Object Container
RTDispatcher
RTMediator
RTOCR
Software-Hardware Interface
RCSM Object Request Broker [R-ORB]

Figure 2: Extensions in the RCSM, shown in Figure 1, for Real-time Context-aware Objects.

In this paper, our approach extends the current RCSM by providing explicit support for context-aware real-time objects and object communication. Figure 2, which ignores the other components in RCSM, shows the major parts of our approach as described below:

a. A Context-enabled Interface Definition Language (CA-IDL) for specifying contexts and associating them with individual real-time object methods.
b. An RCSM-Object Request Broker (R-ORB), which uses dynamically changing contextual data instead of application-initiated actions as criteria to establish impromptu ad hoc communication among devices.
c. Adaptive context-aware object containers that manage method invocation and context-sensitivity of real-time objects.

We will discuss the abovementioned facilities in the following sections. From this point onward, we will refer RCSM with these facilities as *extended RCSM.*

3. Supporting Real-Time Context-Awareness in Interface Definition

To support context-awareness in object interfaces, it is necessary to derive a systematic method to represent contexts. This is accomplished by the following tasks:

1. Categorize contexts in terms of their sources
2. Specification of contexts and their relationships
3. Extend CORBA IDL to specify contexts using the methods from 2.

3.1 Categorization of Context-Sources: Categorization of context-sources is necessary to enable embedded object developers to understand the criticality of different contexts and specify the desired context-sensitivity of the application software. Currently, we divide the contextual data into three different sources:

Network Context: In this category, we mainly consider the routing and transport protocols used in the devices. Although routing protocols usually vary in routing strategies and performance [10,11], a set of common operational and performance parameters, such as change in routing information, are used to incorporate context sensing at the network-layer.

Device Context: In this category, device specific data are collected. The data may include amount of available battery power, the brightness of light surrounding a device, location, time-of-day, and presence of other devices within a specific range.

User-Interaction Context: Depending on the type of a device, user-interaction provides sufficient contextual data, which can be used to guide the communication with other devices. Note that, this information is application-specific and usually originated from different user programs.

3.2 Context-Sensitive Interface Definition: *Context* is considered as any detectable attribute of a device, its interaction with external devices, and/or its surrounding environment. A *context-tuple* is defined as a tuple <ai, aj, ak, ... an, tm> of size *n*, where *n* is the number of unique contextual-data sources present in the device. Each variable *ai* in the tuple represents a value, which is valid for the corresponding type of context. The variable *tm* represents the time of the tuple creation time. Figure 3 shows two examples of

```
//CA-IDL
module NetworkContext {
  short Route-Table-Change;
  short Route-Reply-Message;
  ... ...
};

module DeviceContext {
  long GPS_coordinates_L;
  long GPS_coordinates_La;
  short brightness;
  short adjacent-devices;
};

module UserInteractionContext {

};

//Context-Tuple Definition
Context C;
C.CreateTuple( 3 );
C.AddTuple(0,
  NetworkContext::Route-Table-  Change);
C.AddTuple(1, DeviceContext::brightness);
C.AddTuple(2, DeviceConext::adjacent-
devices);
```

Figure 3: Two Examples of Context-Source and Context-Tuple Definitions in CA-IDL.

context-source and context-tuple definitions in CA-IDL. To be able to specify temporal relationships among multiple context-tuples, several timed-regular expression operators are presented in [2]. These operators are summarized as follows:

- Union: "+"
  Usage: [(C1 + C2) t]
  Either C1 or C2 is true in every discrete time interval t
- Concatenation: "^"
  Usage: [(C1^ C2) t]
  Both C1 and C2 are true in every discrete time interval t
- Repetition: "*"
  Usage: [(C1*) t]
  C1 is repeatedly true in every discrete time interval t
- Precedence: "->"
  Usage: [(C1 -> C2) t]
  C2 becomes true within t time units C1's being true

To illustrate how context-tuples in conjunction with timed-regular operators can be used to define context-awareness for real-time software, we consider a device Di. Di is equipped with five different sensors. These sensors are responsible for location tracking, direction tracking, mobility rate of the device, sensing the amount of light in the surrounding environment, and the amount of available power in the device. A context tuple C1 created at time t can be represented as:

C1: ((x, y), di, m, l, p, t)

Based on the representation, an object o, which is resident in Di, defines a desirable context such as the following - whenever *the mobility rate, m is more than 10 units/time and the light intensity, l is more than 50 units for at least 15 units.* This can be expressed using timed regular expression as follows:

Ca: [((x, y), di, m>10, l>50, p, t)*15]

Here, the x, y, p, and t represent the default values of the location, direction, power level, and tuple creation time, and in this particular case these values are ignored since they are not part of this specific context. Object o defines additional contexts. For example, whenever *the direction of Di is north for 30 units or the direction is south for 10 units.* This context can be expressed as follows:

Cb: [[((x, y), north, m, l, p, t)*30] +
[((x, y), south, m, l, p, t)*10]].

If the second context Cb should occur within 20 time units of the first context Ca, the entire expression can be as follows:

[(Ca -> Cb) 20] or
[[((x, y), di, m>10, l>50, p, t)*15] ->
[[((x, y), north, m, l, p, t)*30] +
[((x, y), south, m, l, p, t)*10]]]

We now describe how to associate context-tuples with the methods of a real-time object. We are developing Context-aware Interface Definition Language (CA-IDL), which is based on CORBA 2.3 Interface Definition Language specification. Additional language constructs are used to associate a method of an object with a particular context-tuple. To accomplish this, the following rules are applied:

- Each method of an interface should be annotated with either [incoming] or [outgoing] tags. An [incoming] tag signifies that the corresponding method should be invoked by an external entity. An [outgoing] tag signifies that the method should be invoked on the remote object.

- Each method of an interface should be annotated with [activate-at-context x] tag, where x represents a context-tuple or timed-regular expression for contexts, as described in Table 1. Note that individual objects can also change the value of x during runtime.

The remaining parts of interface definition are similar to CORBA IDL definition. Whenever the value of a context-tuple becomes true, the corresponding method is invoked.

4. Adaptable Object Container (ADC)

The ADC is responsible for object activation, method invocation, detecting appropriate contexts, and caching outgoing data. In addition, it is responsible for adapting the context-sensitivity of an object to prioritize the execution of critical real-time methods.

4.1 ADC Architecture:

An ADC is associated with one or more objects in an embedded device. The *Real-time Component Framework* (RCF) [15] is the base architecture of ADC. In case of extended RCSM, the RCF is optimized by removing some of its interfaces and *management layer* [15] capabilities.

RTDispatcher: This object is functionally similar to the operations of CORBA Portable Object Adapter (POA) specification.

RTMediator: This object is responsible for caching outgoing object data in case of a network failure due to node mobility or power conservation.

RTObjectContextReflector: This object (RTOCR) tracks the context-tuples for the corresponding object. Figure 4 shows how this object communicates with the remaining parts of extended RCSM during a method invocation following a *context-match* [2] event. The steps are described as follows:

1. New sensor data arrives from sensors to *device-context-reflector*.
2. Sensor data is cached in *context-cache* [optional step].
3. *Device-context-reflector* combines different sensor data and notifies the appropriate *RTOCR* through a software-hardware interface.
4. *RTOCR* notifies the *RTDispatcher, which* in turn activates the appropriate object and method.
5. Communication with the remote object occurs through R-ORBs.
6. In case of connection failure, outgoing data from an object is cached in *RTMediator* for later transmission [optional step].

ADC performs two modes of object activation – local and remote. In local mode, object methods are invoked asynchronously based only on specified contextual conditions. In the remote mode, communication between two or more objects of two

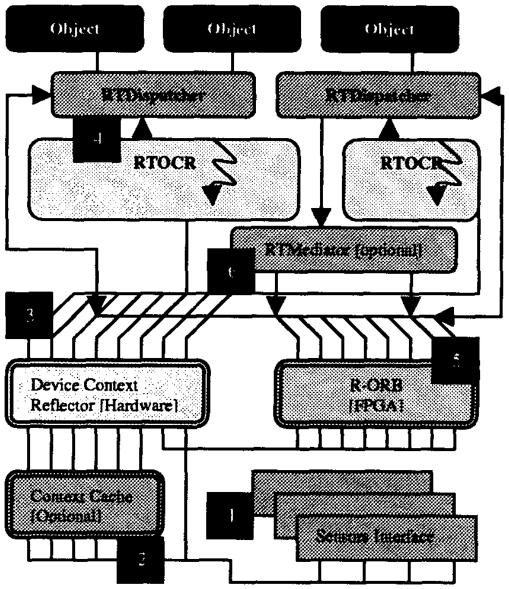

Figure 4: Context-Triggered Object Invocation with Extended RCSM.

different devices occurs after the after the corresponding *context-match* events.

4.2 Adaptation of Context-Sensitivity:

It may be possible that majority of the context-aware methods need to be activated simultaneously in response to several context-match events. This scenario is considered as *context-match burst*. In cases such as this, to guarantee the completion of critical methods, the invocation of some of the less critical methods need to be delayed up to a certain period. In order to systematically analyze this delay characteristic, we define *context-sensitivity* of objects. The context-sensitivity of an object q,

$$CSq=(\#\text{ of context-aware methods of q})/CE * TD, \qquad (1)$$

where CE = # of context-match events over a period t and TD = average delay of all the methods after the corresponding CE is occurred. In case of remote activation mode, (1) becomes:

$$CSq=(\#\text{ of context-aware methods of q})/CIE * TD, \qquad (2)$$

where CIE is the number of occurrence of both *identity-match* [2] and *context-match* events.

From (1), it is clear the context-sensitivity of an object can be either increased or reduced through either the CE or the TD parameter. This is helpful in cases where the context-sensitivity of the objects, invocation frequencies, resource usage, and communication with other objects need to be controlled. Two strategies will be implemented in extended RCSM as follows:

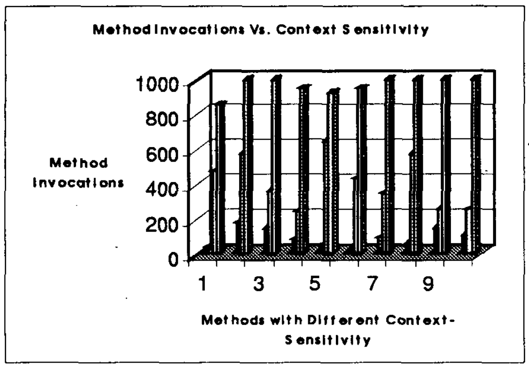

Figure 5: Context-Sensitivity vs. Frequency of Method Invocations.

(a) The value of CE is controlled by restricting or expanding the scope of context-matching criteria. Our experimental results show the effect of this strategy in Figure 5, which illustrates that reduced scope decreases the frequency of method invocation.

(b) By delaying the invocation of the method following a *context-match* event. The amount of delay is computed based on two factors – relative priority of the method and rate of context changes over a period of time. Note that the rate of changes in context-data is a variable, since it also depends on the mobility of a device.

5. Context-Triggered Object Interaction:
One of the characteristics of the extended RCSM is its ability to connect objects of different devices based on the changing conditions of the surrounding environment. This is referred to as context-triggered object activation. In extended RCSM, R-ORB and R-GIOP protocol facilitate context-triggered object activation.

R-ORB and R-GIOP: A device has only one instance of R-ORB. Since it provides performance intensive and reusable functionality, in our architecture the R-ORB will be implemented in FPGA. The R-GIOP is designed to operate over any wireless connection-oriented protocols, such as Bluetooth. R-GIOP is a symmetrical communication protocol.

ORB Connection Establishment: ORB Connection establishment is the process of setting up a communication channel between two remote ORBs. The communication channel is used to exchange information between the objects hosted by the ORBs. In extended RCSM, a connection is established between two R-ORBs based on the following condition:

$$\exists \text{ D1, D2: } \delta(\text{D1} \rightarrow \text{D2}) \wedge (\Delta(\text{D1}) \wedge \Delta(\text{D1})) \wedge (\vartheta(\text{D1}) + (\vartheta(\text{D2}))$$

where D1, D2: Two mobile devices.
δ: Reachability function, which determines if two devices are within the range of each other.
Δ: Context-match function, which determines if the surrounding environment satisfies a specific context specified by a device.
ϑ: Identity-match function, which determines if the objects in two devices are suitable to exchange information with each other.

As it shows from the above expression, an upper bound on the number of connections among the devices is: $\varphi <= \Sigma (\Delta(\text{Di}))$, where Di is the ith device in the environment. The value of φ is the maximum when the following is true:

$$\forall \text{ Di: } (\Delta(\text{D1}) \wedge \Delta(\text{D1})) \wedge (\vartheta(\text{D1}) + (\vartheta(\text{D2})) \quad (3)$$

Scalability Issues: The ORB and the data-marshalling units of a middleware in ubiquitous computing environments must be highly efficient to address different degree of scalability. However, these operations are often regular and often need not be changed. On the other hand, often times sophisticated scalability improvement techniques are not economical in embedded devices due to different constraints. It is shown that reconfigurable processors are useful in computations that are regular [16]. As such, we address the above issue by using dedicated FPGAs to implement R-ORB to provide high-performance and scalability.

6. An Illustrative Example
In this example, we illustrate the techniques used in extended RCSM by showing how context-aware objects in three devices autonomously exchange information under different contextual conditions. The requirements of the system are given below:

R1: The system is a network embedded sensor networks. Three different types of sensor devices (types A, B, and C) are used in the network.
R2: All sensor devices have limited transmission range of 10 meters and usually have limited battery power.
R3: Type B sensors are mobile. As such, the topology of the entire network changes continuously and unpredictably.

R4: Communication among the sensors occur based on the following conditions:
Type A exchanges data with Type B sensors whenever Type B sensors are within the range of Type A. Due to the mobility of Type Bs, the information exchange must occur within 3 seconds of B's presence in A's transmission range. Type C sensors exchange data with Type As based on specific contextual conditions.

The example is divided into two parts. First, we describe individual devices along with built-in objects, their interfaces, and their preferred contexts for activation. The first part also shows how context-aware interfaces can be specified using the CA-IDL of RCSM. Second, we describe two scenarios, and show how respective RCSMs of the devices facilitate autonomous context-based communication.

- Overview of the Devices:

Type A Sensors: Type A sensors are stationary and are responsible for collecting surveillance data about motions in a specific region. These data must be communicated with Type B sensors whenever such a sensor is in range. The object, which is involved in this information exchange, is presented below. Its preferred context is the presence of Sensors Bs in the range of 10 meters.

- Object A-1
  //Name: Surveillance Data Collector
  Interface Definition:
  //ID: 001
  //Define a context variable
  ```
  Context C_A1;
  C_A1.CreateTuple (1);
  C_A1.AddTuple(0,NetworkContext::
  adjacent_devices)
  ```
  //Method 1: activate when sensor B is in range
  ```
  [outgoing] [activate at C_A1]
  exchange_data([out] string data)
  ```
  //ID: 002
  //Define a context variable
  ```
  Context C_A2;
  C_A1.CreateTuple (1);
  C_A1.AddTuple(0,NetworkContext::
  adjacent_devices)
  ```
  //Method 2: activate when sensor C is in range
  ```
  [incoming] [activate at C_A1]
  getnoise_data([in] noise_data)
  ```

The method exchange_data is invoked whenever C_A1 is true and an identity-match event occurs. Here, an identity-match event will occur if the remote object in the adjacent devices also has an ID: 001. Similarly, getnoise_data is invoked when a Type C sensor is in range and an identity-match event occurs to signify that the interfaces 002 match.

Type B Sensors: Type B sensors are mobile. The object interface is shown below:

- Object B-1
  //Name: Surveillance Data Receiver
  //ID: 001
  Interface Definition:
  /*Define a context variable*/
  ```
  Context C_B1;
  C_B1.CreateTuple (1);
  C_B1.AddTuple(0,NetworkContext::
  adjacent_devices)
  ```
  //Method 1: activate when sensor B is in range
  ```
  [incoming] [activate at C_B1]
  exchange_data([in] string data)
  ```

Type C Sensors: Type C sensors stationary and are noise detectors. These sensors notify Type A sensors whenever noise is detected continuously for more than 30 seconds. The object in this sensor is described below:

- Object C-1
  //Name: Noise Detector
  //ID: 002
  Interface Definition:
  /*Define a context variable*/
  ```
  Context C_B1;
  C_B1.CreateTuple (1);
  C_B1.AddTuple(0,DeviceContext::
  Noise)
  Context C_B2;
  C_B2.CreateTuple (1);
  C_B2.AddTuple (0,[(C_B1)*30]);
  ```
  //Method 1: activate when sensor B is in range
  ```
  [outgoing] [activate at C_B2]
  exchange_data([out] string data)
  ```

- **Interaction Scenarios**

Scenario1: A sensor of type B come into the range within the range of a sensor of type A. The following sequence of event occurs:

- Both Sensors A and B detect each other's presence, and establishes a low-level communication channel, using protocols such as Bluetooth.
- The R-ORBs of both sensors establish a connection, and exchanges object identities.
- A *context-match* event and an *identity-match* event occur in both devices. In Sensor A, the *context-match* event occurs since it matches context C_A1, as defined in the interface of Object A-1. Similarly, in Sensor B, matching of context C_B1 triggers a *context-match* event.

- An identity match event occurs in both sensors, since both methods have id of 001.
- The exchange_data methods of both sensors are activated.
- The communication channel is terminated between the respective R-ORBs after the exchange of information is finished.

Scenario2: During Scenario 1, a Type C sensor detects noise for more than 30 seconds. This event triggers a context-match event due to context-tuple C_B2.

- A *context-match* event occurs in Sensor A since Sensor C is also in the range. As in Scenario 1, identity-match occurs. However, depending on the availability of the processing power in the sensors, the corresponding methods are invoked at a later time. This corresponds to increasing the value of TD parameter, thereby reducing the context-sensitivity of the methods.

7. Discussion

In this paper, we have presented how RCSM can be extended to support real-time context-aware applications in ubiquitous computing environment. Future research in this area includes hardware-software co-design of RCSM and adding support for reconfiguration to make RCSM customizable for different embedded devices. The effect of high-frequency changes in contextual data on the deadlines of real-time objects will be explored further to provide real-time scheduling support in RCSM. Additional work needs to be done to refine the R-GIOP protocol to support tolerance for ad hoc object communication in constantly mobile environments. It is also our plan to develop a R-GIOP and CORBA GIOP bridge to enable seamless interoperability with RT-CORBA real-time objects, which are not necessarily context-aware.

Currently, we are implementing the CA-IDL compiler with a mapping to C++. Our current activities also include hardware design of R-ORB, R-GIOP, and the *Device Context-Reflector* using a 20,000 logic-gate Xilinx XC4010XL FPGA device with on-board 32 KB of SRAM, and an 8031 micro-controller. It is our plan to integrate this hardware prototype with software-parts of RCSM using Bluetooth as the underlying communication protocol.

References:

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

[2] S. S. Yau and F. Karim, "Reconfigurable Context-Sensitive Middleware for ADS Applications in Mobile Ad-Hoc Network Environments", to be published *in Proc. 5th IEEE International Symposium on Autonomous Decentralized Systems (ISADS 2001)*, March 2001, Dallas USA.

[3] A. Gokhale and D. Schmidt, "Techniques for Optimizing CORBA Middleware for Distributed Embedded Systems", *Proc. IEEE INFOCOM 99*, http://www.cs.wustl.edu/~schmidt/corba-research-realtime.html.

[4] F. Kon, et al, "Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB", *Proc. IFIP/ACM Int'l Conf. Dist. Systems Platoforms and Open Dist. Processing (Middleware 2000)*, New York, April 2000, http://devius.cs.uiuc.edu/2k/dynamicTAO/.

[5] Bluetooth Consortium, "Bluetooth Protocol Architecture", 1999 http://www.bluetooth.com/.

[6] The Infrared Data Association, "IrCoMM Protocol", http://www.irda.org/.

[7] HomeRF Working Group, "Shared Wireless Access Protocol", http://www.homerf.org/.

[8] F. Bennett et al, "Piconet – Embedded Mobile Networking", *IEEE Personal Communications*, October 1997, http://www.uk.research.att.com/ pen/.

[9] T. Zimmerman, "Personal Area Networks", *IBM Systems Jour.*, Vol. 35, No. 3&4, 1996, pp. 609-617.

[10] E. M. Royer, et al, "A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks", *IEEE Personal Comm.*, April 1999, pp. 46-55.

[11] J. Broch et al, "A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols", *Proc. 4th Int'l Conf. Mobile Computing and Networking*, October 1998, pp. 85-97.

[12] S. S. Yau, "Achieving Quality Software Development for Distributed Environments", *Proc. 1st Asia-Pacific Conf. on Quality Software*, October 2000, pp. 169-178.

[13] IETF, "Mobile Ad Hoc Networking (MANET)", March 1998, http://www.ietf.org/1id-abstracts.html.

[14] M. Haahr, Raymond Cunningham and Vinny Cahill, "Supporting CORBA Applications in a Mobile Environment", *Proc. 5th ACM/IEEE Int'l Conf. Mobile Computing and Networking (MobiCom 99)*, August 1999, pp. 36-47.

[15] S. S. Yau and F. Karim, "Component Customization for Object-Oriented Distributed Real-time Software Development", *Proc. 3rd IEEE Int'l Symp. Object-Oriented Real-time Dist. Computing (ISORC 2000)*, March 2000, pp. 156-163.

[16] A. DeHon, "The Density Advantage of Configurable Computing, *Computer*, Vol. 33, No. 4, April 2000, pp. 41-49.