Real-Time Systems, 26, 29–61, 2004

An Adaptive Middleware for Context-Sensitive Communications for Real-Time Applications in Ubiquitous Computing Environments

STEPHEN S. YAU yau@asu.edu
Computer Science and Engineering Department, Arizona State University, Tempe, AZ 85287, USA

FARIAZ KARIM fariaz.karim@intel.com
Intel Corporation, Hillsboro, OR, 97124, USA

Abstract. Context-sensitivity is an important expected capability in applications in ubiquitous computing (ubicomp) environments. These applications need to use different contextual information from the user, host device, on board sensors, network, and the ambient environments to systematically adapt their actions. In addition, some context-sensitive applications may use specific contextual conditions to trigger impromptu and possibly short-lived interactions with applications in other devices. This property, referred to as *context-sensitive* or *context-aware communications*, allows applications to form short-range mobile *ad hoc* networks consisting of mobile and stationary devices, sensors, and other computing resources. Real-time applications, especially those having reactive behavior, running on embedded devices and requiring context-sensitive communications support, pose new challenges related to systematic representation of specific contexts, associations of contexts with real-time actions, timely context data collection and propagation, and transparent context-sensitive connection establishment. An object-based middleware can be effective to meet these challenges if such a middleware can provide a well-defined development framework as well as lightweight runtime services. In this paper, an adaptive and object-based middleware, called reconfigurable context-sensitive middleware (RCSM) is presented to facilitate context-sensitive communications in ubicomp environments. To facilitates context-sensitive communications, RCSM provides a context-aware interface definition language for specifying context-sensitive interfaces of real-time objects, an object container framework for generating interfaces-specific context-analyzers, and a context-sensitive object request broker for context-sensitive object discovery and impromptu connection management. RCSM is adaptive in the sense that depending on the context-sensitive behavior of the applications, it adapts its object discovery and connection management mechanisms.

Keywords: reconfigurable context-sensitive middleware, ubiquitous computing environments, real-time context-sensitive applications, context-sensitive communication, context-sensitive object discovery, adaptive middleware, mobile *ad hoc* networks, object-based middleware, context-sensitive object request broker

1. Introduction

Ubiquitous computing (ubicomp) environments (Weiser, 1991, 1993) focus on integrating computers with the physical environments to make computing and communication essentially transparent. Devices that operate in these environments are usually embedded and use low-power wireless communication means, form numerous webs of mobile *ad hoc* wireless networks or MANET (IETF, 2001) to exchange information, and react in a transparent fashion. In ubicomp environments, the topologies of the networks are dynamic due to arbitrary node mobility, and the networks usually contain no dedicated network connectivity devices. Applications in ubiquitous environments should address the following characteristics (Abowd and Mynatt, 2000; Oxygen, 2000; Yau and Karim, 2001a):

- *Context sensitivity*: Applications use various data about the surrounding environment and the available resources to adapt their behavior and interactions.
- *Context-sensitive interactions*: Applications software in different devices exchange data with each other through communication channels that are instantaneously established, short lived, and terminated due to changing contexts and node mobility.
- *Intelligent human computer interaction*: Applications software provides richer and natural interfaces to support common forms of human expressions, such as speech, hand gestures, etc.

While the characteristics related to human computer interaction are essentially application-specific, characteristics related to the other two kinds (i.e., 1 and 2) specify the need for system support for both development and runtime services for applications in ubicomp environments. In case of context-sensitivity, it is necessary to provide customized capabilities for applications, especially for real-time applications, where detecting and analyzing different context must be accomplished in a timely fashion. For context-sensitive communications, the runtime services of a device must be able to detect, establish, and terminate communication channels autonomously as new devices come in contact or existing devices move away. These requirements imply that the underlying system services must themselves be context-sensitive, efficient, and address the vast heterogeneity of devices in ubicomp environments. In this respect, middleware-oriented approaches can be very effective if they can reduce the effort required to develop ubicomp software and provide appropriate runtime services for applications with the above-mentioned characteristics, in addition to providing the typical middleware services, such as interoperability, transparency, object-orientation, etc. (OMG, 2001) However, current middleware solutions are not adequate to support context-sensitivity for real-time software in ubicomp environments because they are designed based on the assumption that the underlying network is fixed and application software are not context-sensitive. In addition, these middleware implementations mainly use client–server interaction semantics, which are not the only type of interaction in ubicomp environments. Moreover, most currently available middleware for mobile networks, as we will summarize in a later section, do not make use of the context, which is often one of the most important properties of applications in ubicomp environments (Abowd and Mynatt, 2000).

In this paper, we will present an adaptive and object-based middleware, called reconfigurable context-sensitive middleware (RCSM) to facilitate context-sensitive communications in ubicomp environments. To achieve this, RCSM provides a context-aware interface definition language for specifying context-sensitive interfaces of real-time objects, an object container framework for generating interfaces-specific context-analyzers, and a context-sensitive object request broker for context-sensitive service discovery and impromptu connection management. RCSM is adaptive in the sense that depending on the context-sensitive behavior of the applications, it adapts its object discovery and connection management mechanisms. The architecture of RCSM consists of both application-specific and application-independent components. Application-specific components are implemented in software while the application-independent components can be realized in software or be implemented in reconfigurable hardware for providing better flexibility and performance.

2. Background and Related Work

2.1. Background

2.1.1. Ubicomp Environments: Ubicomp Environments

The concept of ubicomp (Weiser, 1991, 1993) focuses on integrating computers with the physical environment, making computing and communication essentially transparent to the users. An interesting perspective is found in (DARPA, 1999), which differentiates ubicomp environment from current computing infrastructures as follows: Current computing infrastructure is a ''one size fits all'' model, while ubicomp makes the user the center of computing, thereby dynamically adapting the computing model to individual users' needs and actions. Another goal of ubicomp is to make the computing experience natural and everywhere to such a degree that the enabling technologies become essentially invisible. Popular terms associated with ubicomp include pervasive computing (IBM, 2001), invisible computing (Aura, 2000) etc. A ubicomp environment can be thought of as a collection of connected computing nodes, such as embedded, wearable, handheld computers, PCs, and workstations, that transparently cooperate with each other to make computing and communications available to the user virtually everywhere with a user-centric model. Construction of this type of environment now seems possible due to rapid progress in inexpensive, short-range, and low-power wireless communication hardware and emerging low-power networking standards, such as Bluetooth, IrDA, and IEEE, 802.11 (Bluetooth, 1999; IEEE, 1997; IrDA, 2001).

2.1.2. Mobile Ad Hoc Networks (MANET)

A MANET (IETF, 2001) is a collection of connected autonomous mobile nodes, such as wearables, handhelds, and other embedded devices with the following characteristics:

- the nodes are free to move arbitrarily (e.g., a wearable device moves with a living carrier);
- usually have bandwidth and energy constraints; and

- the topologies in MANETs are dynamic, and MANETs usually contain no dedicated network connectivity devices, such as routers or a Global System for Mobile Communications (GSM) base station, to assist the nodes in data communication.

These characteristics distinguish MANETs from other mobile networks with fixed routing infrastructure, often known as infrastructure-based mobile networks, such as personal communication systems (PCS) and GSM cellular technology (Rappaport, 1995). Applications of MANETs include mobile command and control centers, public-meeting places, and especially the cases where a fixed network infrastructure is neither available nor feasible, and where rapid communication setup and termination are necessary. 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. Devices operating in MANETs have memory and processing constraints and usually have close coupling with specific hardware components. Most of the research efforts in MANET have been in devising routing techniques (Broch et al., 1998; Gupta, 1999; Lee, 1999, 2000; Royer and Toh, 1999; Vaidya, 2001). On the other hand, evolving standards and effort on low-power networking, such as Bluetooth (Bluetooth, 1999), IrDA (IrDA, 2001), HomeRF (HomeRF, 2001), IEEE 802.11 (IEEE, 1997), IEEE 802.15 (IEEE, 2001), Pen (Bennett et al., 1997), PicoRadio (Rabaey et al., 2000), are now enabling significant advantage to construct and experiment with various types of ubicomp applications. The standards usually vary in terms of their support for voice and data, the layers of protocols supported, and most importantly the power and cost of the radio hardware. For example, the goal of PicoRadio (Rabaey et al., 2000) is to devise a very cheap radio technology (50 cents per device as oppose to \$5–10 for a Bluetooth device) and ultra-low power consumption (100 μW for PicoRadio as oppose to 100 μW for Bluetooth) by trading-off the transmission range (2 m in contrast with Bluetooth's 10 μ). On the other hand, IEEE 802.11b, which is by far the most widely used wireless network technology, provides a typical range of up to 100–200 m with a data rate of 11 Mbps. Another standard, IEEE 802.11a, provides 54 Mbps in a higher frequency. However, the high performance causes IEEE 802.11-based products to cost more and to consume more power.

2.1.3. Middleware

In general, middleware is a collection of cooperating components that strive to achieve the following goals (Bernstein, 1996; Schreiber, 1995):

- To increase the degree of interoperability among software packages, which are distributed and are developed using different languages, operating systems, and hardware platforms.

- To facilitate the interactions among distributed applications by elevating the programming abstraction and forcing a clear separation between interface and implementation.

2.1.4. Context, Context-Sensitivity, and Context-Sensitive Applications

DEFINITION 1 *The context with respect to a particular device is any detectable and relevant attribute related to the device, and its interaction with other devices for the application software of the device to take different actions adaptively in different contexts (Yau and Karim, 2001a).*

For example, a device's current geographical location, the history of a device's interactions with other devices, and its available battery power are some of its contexts, assuming that these attributes are of interest to some application software installed in the same device. A more general definition of context can be found in (Dey, 2001), where context is any information that can be used to characterize the situation of an entity. It is important to note that the concept of context is the building block for representing situation, which allows ubiquitous applications to act in a more intelligent manner by capturing the patterns in context and past actions over a period of time.

DEFINITION 2 *The context-sensitivity of an entity is the ability of the entity to detect its current context and changes in any contextual data, and respond in a well-specified fashion.*

DEFINITION 3 *Context-sensitive applications are applications that are context-sensitive and are able to adaptively take different actions in different contexts.*

Examples of an entity in Definition 2 are application objects, middleware implementation, and devices. Examples of context-sensitive applications can be found in (Chen, 2001; Marmasse and Schmandt, 2000; Pascoe, 1998). Context-sensitive real-time software can be characterized as a particular kind of context-sensitive software, which detects and analyzes different contexts to manage the invocation and scheduling of its time critical tasks.

2.2. Related Work

Although little work has been done on middleware for ubicomp environments, it is necessary to know the existing work on middleware for both enterprise and mobile networks due to their common objectives for facilitating the development and runtime operations of various applications. The existing middleware for enterprise and mobile networks can be divided into commercially available middleware implementations with corresponding industry standards, and openly available specialized laboratory-version middleware. For the enterprise networks, available middleware implementations are mostly based on industry standards and their variations, such as CORBA (OMG, 2000), COM (COM, 1995), Real-time CORBA (OMG, 2001), TINA-C (TINA-C, 1998). In addition, openly available laboratory-version middleware, such as TAO (Gokhale and Schmidt, 1999; Loyall, 1999) and dynamicTAO (Kon et al., 2000), have been developed for real-time and resource-constrained systems. In the remainder of this section, we will summarize the existing middleware for mobile networks due to the dependence of ubicomp applications on this type of networks. This type of middleware is currently available as laboratory-versions, not as commercial products.

2.2.1. Middleware for Mobile Networks

The Mobiware (Campbell, 1998) project at Columbia University provides the facilities for managing an open, active, and adaptive mobile network. Mobiware accomplishes its goal by taking advantage of a CORBA-based architecture (OMG, 2001) and using different adaptive algorithms as Java objects, which can be injected dynamically into mobile devices, access points, and mobile-capable network switches or routers. Thus, it enables the addition of value-added quality of service (QoS). Applications of Mobiware include QoS-controlled handoff management and flow bundling during the mobility of different devices. Mobiware can be considered as a solution for developing applications that manage the infrastructure of a mobile network, as opposed to supporting the end user applications.

The Architecture for Location Independent CORBA Environments (ALICE) (Haahr et al., 1999) project at Trinity College modifies the CORBA architecture, by introducing a mobility layer between transport layer and the CORBA IIOP layer to provide support for both mobile clients and mobile servers. The limitation of ALICE, however, is its assumptions on the fixed mobile gateways and client–server semantics that make it fairly restricted in applications for mobile *ad hoc* networks.

The LIME middleware (Murphy et al., 2001) at Washington University adopts a coordination perspective based on shared tuple space model. The programming model offered by LIME views mobility as transparent changes in the content of the tuple space. This model easily supports interactions among mobile devices in mobile *ad hoc* networks since the tuple-space model supports location transparency and disconnected operations. In addition, LIME supports context-sensitivity in a limited sense at the application level, but only treats contexts based on the data stored in the tuple spaces. Therefore, it ignores the state of the device and the surrounding environment as parts of the overall contextual condition.

The project of XMIDDLE (Mascolo et al., 2002) at University College, London develops a middleware that is specifically focused on sharing and replication of documents in mobile *ad hoc* networks. Although XMIDDLE relies on a coordination model, it improves the mechanism by taking advantage of hierarchical structure of extensible markup language (XML) (W3C, 2001) documents. The middleware further supports versioning and merging of documents in application-specific fashion.

The QoS-Aware Middleware (Nahrstedt et al., 2001) and iMAQ (Chen et al., 2002) projects at University of Illinois Urbana-Champaign provide cross-layer middleware frameworks to support QoS in mobile *ad hoc* networks. The services range from QoS specification, translation, and runtime adaptation. The adaptation policies are based on a fuzzy control model. The cross-layer architecture spans across route computation, forwarding, group communication, data accessibility service, and location management.

TSPACES (TSAPCES, 1998) by IBM Research focuses on a Java-based communication middleware based on the concept of tuple-spaces. It provides asynchronous messaging-based communication facilities without any explicit support for context-awareness. BlueDrekar (BlueDrekar, 2000), also by IBM Research, is a middleware to provide a protocol layer and a set of APIs to enable communication between two Bluetooth-equipped devices.

Overall, existing middleware technology has the following difficulties to support real-time application software in ubicomp environments:

- Currently available industry standard middleware implementations are designed based on the assumption that the underlying network is stable. In addition, these middleware implementations mainly use client–server interaction semantics, which are not usually suitable in ubicomp environments.

- Most middleware for mobile networks do not make use of the different context, which is often the most important aspect in ubicomp environment. Moreover, no mechanisms yet exist to provide a direction correlation between context and real-time operations of application objects.

- Many existing laboratory versions of middleware have their own unique architectures and interaction semantics. This heterogeneity in their architectures may defeat the main purpose of actually using them for real-world applications, since it becomes difficult to achieve interoperability among different middleware architectures.

- None of the existing middleware implementations directly address the major aspects of ubicomp environments, such as context-sensitivity, and context-sensitive communication.

3. Expected Capabilities of an Adaptive Middleware for Ubicomp Environments

Based on discussions in the previous section, we can summarize what characteristics an adaptive middleware to facilitate context-sensitive communications in ubicomp environments must or should have:

(S1) Uniform development support: Almost all the commonly used programming languages that exist today do not have any basic support for expressing context-awareness. Even if context-aware languages will exist in the future, the support for expressing context-awareness on a conceptual level most likely to be different across different languages. As such, a middleware needs to provide a uniform and common way to express context-awareness of application software without restricting the application software developers to use a specific programming language, operating system, or programming environment.

(S2) Application-specific context acquisition, analysis, and detection: A middleware should provide a uniform and platform independent interface for various applications to

express their needs for different contexts without knowing how the context data is actually acquired.

(S3) Application- and device-independent context acquisition and propagation: Most context data are usually acquired from low-level sources, such as sensors, networks, and device hardware. To facilitate reusability of application software, a middleware should provide a facility for acquiring raw context data from various sources, package these raw data into structured information, and propagate this information to different applications according to their context-sensitivity properties. This will avoid the need for writing application- or device-specific code for performing these low-level operations.

(S4) Context-triggered action: Context-sensitive application software programs usually need to decide what actions to take based on the current context. The action may involve adapting to the new environment, notifying the user, communicating with another device to exchange information, or any other task. A middleware should provide the facilities for application software programs to define such context-triggered actions so that these actions are transparently invoked whenever the corresponding contexts are valid.

(S5) Transparent support for spontaneous and ad hoc communication: A middleware should be able to abstract the details of *ad hoc* communications from the applications to facilitate interoperability independent of network type. The topologies in mobile *ad hoc* networks change dynamically and devices may not know each other *a priori*. On the other hand, a device in a mobile *ad hoc* network may connect to a previously known computer (e.g., a file or a web server) in a wired network. Thus, a middleware should facilitate a transparent communication model so that application software can flexibly interact with each other in different network environments. Also, this middleware should proactively discover new devices and its functionality, establish new communication links, and notify the application layer whenever a ''compatible'' device is found.

(S1) to (S5) correspond to the capabilities required in an adaptive middleware for ubicomp environments. In addition, a middleware for ubicomp environment should also have some other desirable capabilities, such as reconfigurability, lightweight design, and low power consumption, to improve its overall usability for energy-constrained devices. In addition, in order to make the middleware expandable in different application domains, it is desirable to have a reusable framework in the middleware to facilitate the generation of higher-level domain-specific services, such as group communication and security.

4. Our Approach to Developing the Adaptive Middleware

In order to have an adaptive middleware with the afore-mentioned characteristics, we have developed the RCSM (Yau, 2001a; Yau, 2002) to facilitate the development and runtime operations application software that requires context-awareness and/or spontaneous and *ad hoc* communications. In addition, we support these two characteristics in an integrated way to provide context-sensitive communications to the

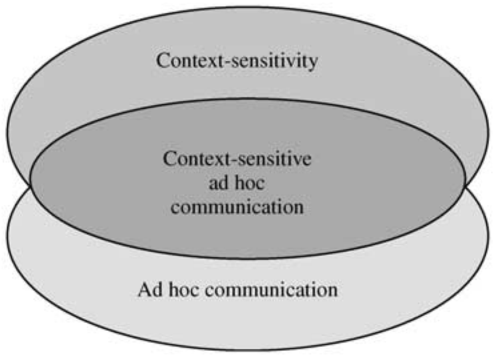

Figure 1. Application software with properties 1–3 facilitated by our RCSM.

application software. This concept is shown in Figure 1, where context-sensitive communications represent the intersection of context-sensitivity and *ad hoc* communication.

We summarize the major capabilities of RCSM in the following subsections. The capabilities presented in Sections 4.1–4.3 address the desired characteristic S1 as mentioned in the previous section. Section 4.4 addresses the characteristics S2 and S4. S3 and S5 are addressed in Sections 4.5 and 4.6, respectively.

4.1. Compliance with OMG's Object Management Architecture (OMA)

We designed RCSM to be compliant with the OMA framework to make it both practical and easy to use with other industry standard middleware implementations. This makes RCSM possibly the first CORBA-compliant middleware for ubicomp environments. Similar to OMA and as shown in Figure 2(a) and (b) we represent user-level application software as application objects. However, unlike OMA, there are two types of application objects in RCSM: context-sensitive and client–server. Figure 2(a) and (b) also illustrate

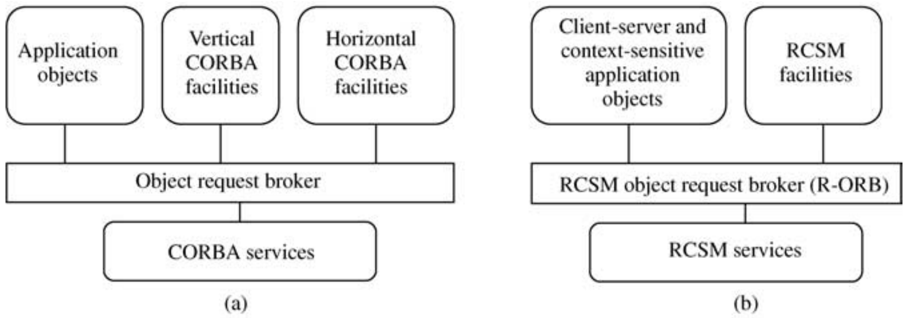
Application objects
Vertical CORBA facilities
Horizontal CORBA facilities
Object request broker
CORBA services
(a)
Client-server and context-sensitive application objects
RCSM facilities
RCSM object request broker (R-ORB)
RCSM services
(b)

Figure 2. (a) Object management group's object management architecture (OMA); (b) RCSM's OMA-compliant architecture.

that similar to OMA's object request broker, the RCSM's object request broker (R-ORB) is the component that enables application objects, implemented in different languages, to communicate in a distributed environment consisting of heterogeneous networks and platforms. In addition to providing client–server communications, R-ORB provides context-sensitive communications to the application objects. RCSM's Context-aware interface definition language (CA-IDL), which is based on CORBA's interface definition language (IDL), can be used to develop application objects by separating their interfaces from their implementations. We also provide the necessary capabilities in RCSM to develop and deploy additional services and facilities for domain-specific use of RCSM.

4.2. Context-Sensitive Objects as Building Blocks for Context-Sensitive Application Software

Similar to mature middleware standards and prototypes, such as CORBA, COM, and TAO (COM, 1995; Gokhale, 1999; OMG, 2000) for fixed networks, RCSM provides an object-based framework for supporting context-sensitive applications. In addition to supporting client and server objects, RCSM provides context-sensitive objects as the building blocks of context-sensitive application software.

DEFINITION 4 *A context-sensitive object is an object that consists of two parts: a context-sensitive interface and a context-independent implementation. The interface encapsulates the description of the ''context-awareness'' of the application, while the implementation remains ''context-free''.*

Figure 3 provides a conceptual picture of a context-sensitive object. Representing application software as context-sensitive objects allows developers to easily express the ''context-awareness'' property within the realm of object-oriented software development methodologies. As illustrated in Figure 3, taking an object-based approach in RCSM presents additional leverages beyond the benefits provided by the simple object-

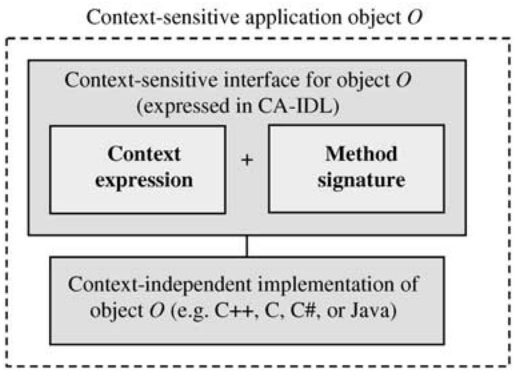
Context-sensitive application object O
Context-sensitive interface for object O (expressed in CA-IDL)
Context expression
+
Method signature
Context-independent implementation of object O (e.g. C++, C, C#, or Java)

Figure 3. A Context-sensitive object O in RCSM.

orientation. The important benefit of this method is that the implementation is completely isolated from the context specification, i.e., it is context-independent. Using the context-sensitive interfaces, RCSM determines which type of context to monitor, how often to monitor, and which method of the objects to activate in the current context. On the other hand, the developer only needs to focus on implementing the methods of the object in his/her favorite language without worrying about context monitoring, detection, and analysis.

4.3. CA-IDL for Expressing Object-Specific Context-Awareness

We have developed CA-IDL to allow application developers to specify context-sensitive interfaces. The syntax of CA-IDL is similar to CORBA's IDL (OMG, 2000), but CA-IDL has additional language constructs to define context expressions as shown in Figure 2. We have made CA-IDL based on CORBA since CORBA's IDL already provides a mature and industry standard interface specification mechanism. This makes our RCSM easy to use for developers who are already familiar with CORBA- (or COM)-based application software programming.

4.4. Interface-Specific Context-Analyzers for Providing Context-Sensitivity

An important feature of RCSM is that the difficult tasks related to context acquisition, monitoring, and detection are done in the middleware level, rather than in the application software level. This greatly simplifies the application software development. However, since the context-awareness requirements usually vary across applications, RCSM provides adaptive object containers (ADCs), which are customly-generated interface-specific context analyzer components. As shown in Figure 4, these components are interface-specific because each interface of a context-sensitive object is, in fact, expresses

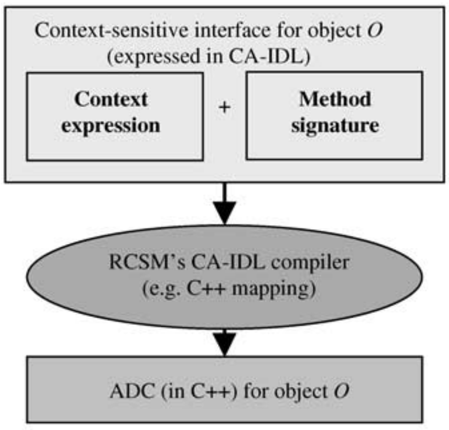

Figure 4. Interface-specific ADCs for providing object-specific context-awareness.

the context-awareness of that object. Based on each interface description, RCSM's CA-IDL compiler generates a custom-made ADC tailored for a particular context-sensitive interface. During runtime, an ADC communicates with other RCSM components to acquire context data and performs periodic context analysis according to the interface specification. It also communicates with the object implementation to invoke different methods whenever suitable contexts are detected as a result of the analysis. Instead of being static context-processing mechanisms, the type of ADCs that can be generated is virtually unlimited. For a different requirement, the application developer just needs to specify a different interface in CA-IDL and compile it to generate a new ADC.

DEFINITION 5 *Context-triggered object activation is a type of object activation mechanism where an ADC activates a context-sensitive object whenever the ADC detects that it is suitable to activate the object in the current context.*

4.5. Application- and Device-Independent Context Acquisition and Processing Framework

According to Definition 1, contexts can cover a wide range of information that could be analyzed by the ADCs on behalf of the objects. Realistically, this information is limited to the capabilities of the hardware (e.g., sensors) and the low-level operating system support. Moreover, these capabilities vary across devices. We have designed a device-independent context acquisition and processing framework (R-CAP) in RCSM to perform low-level context monitoring and acquisition using the available underlying system services. As shown in Figure 5, the framework is divided into several parts, and it facilitates the construction of higher-levels (Levels 2 to *N*) context processing components.

Context-interest registration: Since during runtime ADCs need latest context information, each ADC registers its interest in different context data according to the corresponding CA-IDL specification. The registration is shown as the dotted arrowhead in Figure 5. Based on this registration, RCSM's R-CAP framework decides what information needs to be monitored and propagated to an ADC during runtime.

Context acquisition from local sources: Some contexts can be acquired from the device itself, and without relying on other computing resources. For this type of contexts, the R-CAP periodically follows a deterministic context acquisition procedure that guarantees availability of latest context data to the ADCs. Examples of such context types are current battery power of the device, the list of currently opened programs, etc.

Context acquisition from remote sources: Some contexts, however, cannot be acquired from the local device, and RCSM must rely on remote computing resources. For this type of contexts, R-CAP uses a different context acquisition procedure that collects context data whenever such remote sources are available. Figure 5 illustrates the remote context sources as the dotted rectangle. An example of such context types is the current location of a device where the location beacon must be heard from a beaconing server. Since remote context sources may not be available all the time, R-CAP, in this case, cannot guarantee the availability of latest contexts that need to be acquired remotely.

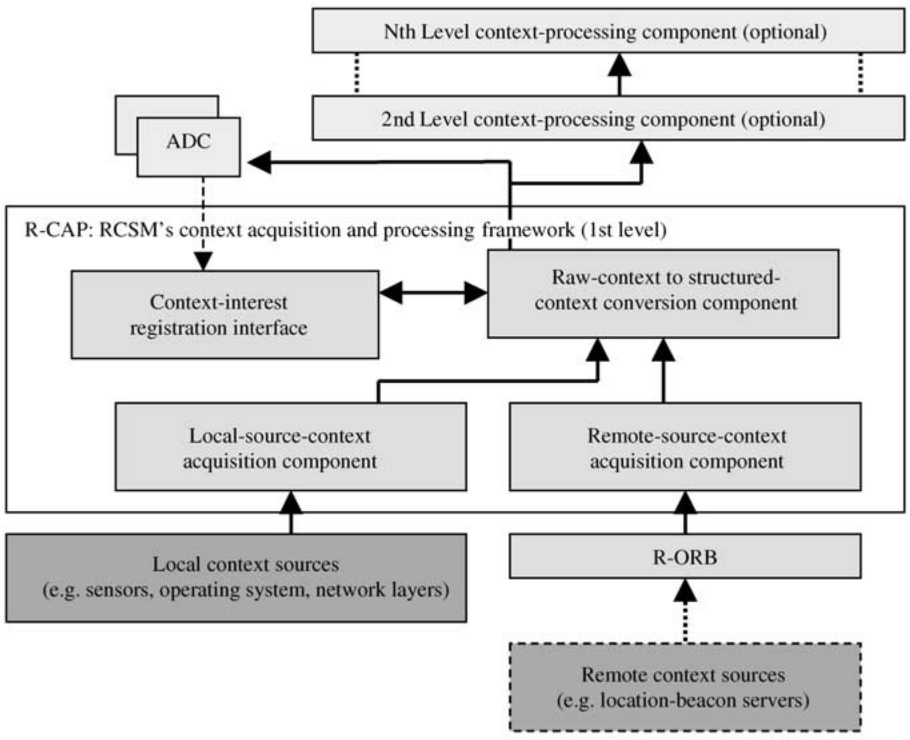
Nth Level context-processing component (optional)
2nd Level context-processing component (optional)
ADC
R-CAP: RCSM's context acquisition and processing framework (1st level)
Context-interest registration interface
Raw-context to structured-context conversion component
Local-source-context acquisition component
Remote-source-context acquisition component
Local context sources (e.g. sensors, operating system, network layers)
R-ORB
Remote context sources (e.g. location-beacon servers)

Figure 5. Components of RCSM's context acquisition and processing framework, R-CAP.

Raw context data to structured context data conversion: Periodically, after every acquisition process, R-CAP converts the raw context data to higher-level structured context data according to the need of the ADCs. As the solid arrowheads in Figure 5 illustrate, the raw context data from the context acquisition components are periodically propagated to the raw- to structured-context conversion component, which then propagates the structured context data to the ADCs.

4.6. RCSM Object Request Broker for Context-Sensitive and Ad Hoc Communications

We have designed the R-ORB as the key mechanism in RCSM to provide context-sensitive and *ad hoc* communications to the application software. R-ORB hides the intricacies of mobile *ad hoc* networking. As shown in Figure 6, it also performs remote device and object discovery on behalf of the application, and communicates with the ADCs whenever devices with appropriate services are found. Figure 6 also illustrates how the component-based architecture of RCSM facilitates this dynamic cooperation between R-ORB and ADCs to allow *ad hoc* and application-transparent exchange of information between a pair of distributed context-sensitive objects. In addition to

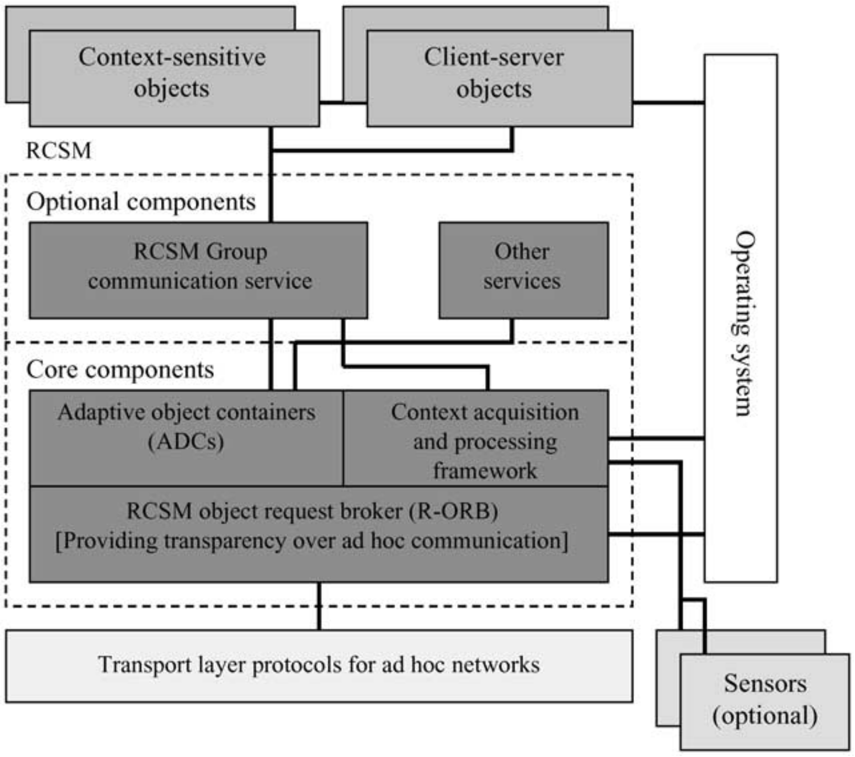
Context-sensitive objects
Client-server objects
RCSM
Optional components
RCSM Group communication service
Other services
Core components
Adaptive object containers (ADCs)
Context acquisition and processing framework
RCSM object request broker (R-ORB) [Providing transparency over ad hoc communication]
Operating system
Transport layer protocols for ad hoc networks
Sensors (optional)

Figure 6. RCSM's component hierarchy.

providing client–server communication facilities, RCSM provides context-sensitive communication, and provides the communication facilities for R-CAP's remote context acquisition.

DEFINITION 6 *Context-sensitive communication is a type of communication, where the communication channel is established between a pair of devices based on some specific contexts.*

DEFINITION 7 *Context-sensitive inter-object communication is a type of context-sensitive communication between two distributed context-sensitive objects, such that these objects are compatible to communicate with each other and both objects can be activated in the current context according to their context-sensitive interfaces.*

In Definition 7, by compatibility we mean that both objects are able to meaningfully exchange data with each other. The compatibility between two objects can be determined through the matching of the objects' interface signatures, including the number and types of parameters, or other application-specific criteria, such as the matching of radio-frequency identifiers, security attributes, etc.

DEFINITION 8 *A context-triggered communication channel between two devices is a type of communication channel, setup by the R-ORBs in these devices to facilitate the context-sensitive inter-object communication between a pair of objects.*

According to Definition 8, an R-ORB may need to simultaneously manage multiple context-triggered communication channels. The exact number depends on the number of context-sensitive objects and other devices in the network.

We use the R-ORB-oriented approach in developing RCSM, because providing only a context-sensitive communication protocol is not sufficient in a middleware. We still need a good ''packaging'' solution to uniformly ease both the development and runtime aspects between the context-processing layer (i.e., the ADCs) and the communication layer (i.e., context-triggered communication mechanism). We have chosen the CORBA ORB-oriented approach because the ORB functions as a single gathering point for (a) appropriately isolating the intricacies of various transport protocols from the application objects by providing platform independent inter-object communication facilities, (b) providing uniform interfaces to deploy and use various distributed services in the middleware for enhancing the capabilities of the application objects, and (c) providing reusable mechanisms to facilitate remote object discovery, object registration, object activation and method invocation, data marshalling, and communications over distributed environments. Without an ORB, it is difficult and cumbersome to provide these capabilities in the transport layer, in the operating systems, or in the application layer. Finally, R-ORB allows us to satisfy (S5) as described in Section 3.

Figure 7 shows how RCSM's capabilities can be used to facilitate the development and runtime operations of real-time application software for ubicomp environments. Each arrowhead in Figure 7 corresponds to a step. Step (R1) corresponds to the specification of

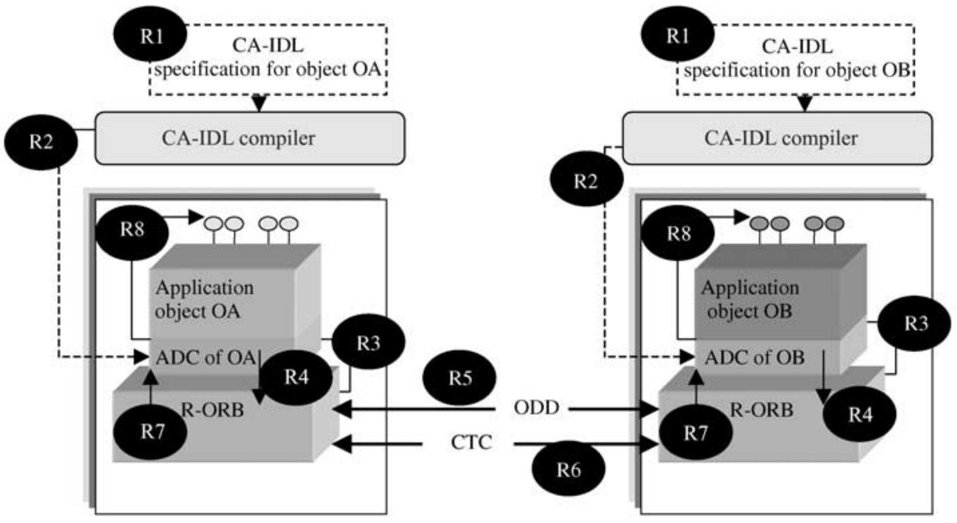

Figure 7. RCSM's overall functionality (ODD: object discovery message; CTC: context-triggered communication channel).

the interfaces of some context-sensitive objects. Step (R2) corresponds to the generation of interface-specific ADCs based on Step (R1). Step (R3) corresponds to R-CAP's context propagation to the ADCs. Step (R4) illustrates an ADC's notification to the R-ORB to indicate that the ADC has detected the suitability of object activation in the current context. Steps (R5) and (R6) correspond to R-ORB's remote object discovery and context-triggered communication channel establishment processes. In Step (R7), R-ORB notifies the ADCs that it has found a compatible remote object for the local object. In response to this notification, the ADC in Step (8) activates the object.

We will provide a detailed overview of RCSM's development and runtime supports for real-time software in Section 5. We will present the details related to Steps (R1), (R2), (R4), and (R8) in Sections 6 and 7. In Section 8, we will discuss the details related to Steps (R5), (R6), and (R7). We will conclude the paper by providing an illustrative example. In this paper, we do not provide the details corresponding Step (R3).

5. Support for Development and Runtime Operations of Real-Time Ubicomp Application Software

RCSM's application software development framework is similar to CORBA (or COM's) client–server development process. In addition to client–server applications, RCSM facilitates the development of context-sensitive application object as described before. As shown in Figure 8(a), an application developer begins by specifying the interfaces of context-sensitive application objects. A procedure for identifying context-sensitive

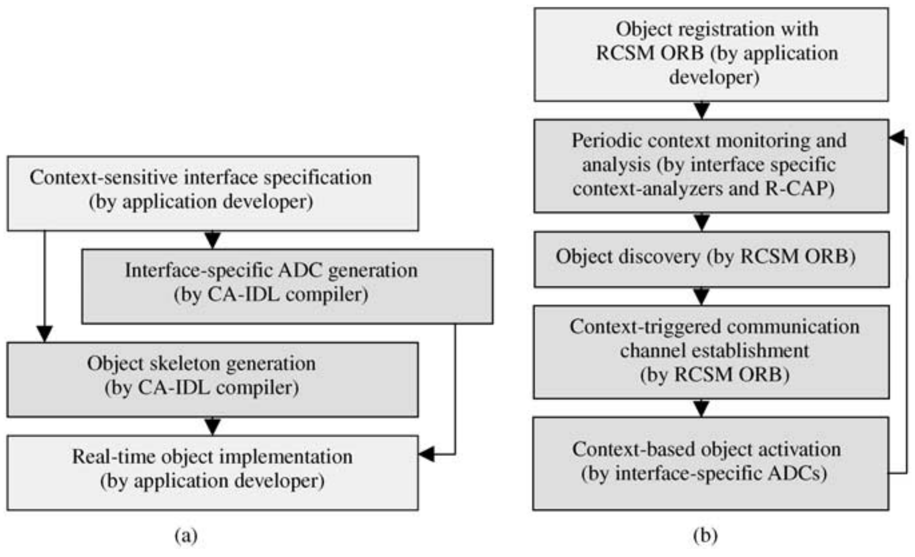
Context-sensitive interface specification (by application developer)
Interface-specific ADC generation (by CA-IDL compiler)
Object skeleton generation (by CA-IDL compiler)
Real-time object implementation (by application developer)
(a)
Object registration with RCSM ORB (by application developer)
Periodic context monitoring and analysis (by interface specific context-analyzers and R-CAP)
Object discovery (by RCSM ORB)
Context-triggered communication channel establishment (by RCSM ORB)
Context-based object activation (by interface-specific ADCs)
(b)

Figure 8. (a) Context-sensitive real-time software development support in RCSM; (b) runtime services of RCSM to support context-sensitive communications.

application objects from the application requirements was described in (Yau, 2001b). The interface specification is then fed into RCSM's CA-IDL compiler, which generates a pair of interface-specific ADC and object skeleton. The developer then implements the functionality by extending the object skeleton. The interface-specific ADC embeds the mechanism to provide the necessary context-awareness support to the application object. After the developer completes the implementation, the application object then becomes ready for deployment. In case of client–server type application objects, CA-IDL generates a simpler version of ADC, which functions similar to a CORBA's portable object adapter (OMG, 2001).

An application developer can deploy an application object by registering the object with the R-ORB. As shown in Figure 8(b), after the registration, RCSM provides four main services to the application objects during runtime. The interface-specific ADC and RCSM's R-CAP framework cooperatively provide the necessary context-awareness support to the application object. On the other hand, R-ORB provides object discovery and communication services. In case the application object is of type client–server, R-ORB simply functions as a client–server ORB to propagate and accept client–server messages to/from remote devices. However, if the application object is context-sensitive, R-ORB uses its context-triggered object discovery and channel establishment procedure to spontaneously establish communications with compatible application objects in remote devices.

6. Specification of Context-Sensitive Interfaces for Real-Time Context-Sensitive Objects

In this section, we discuss how an application developer can perform Step (R1) by specifying the context-sensitive interfaces for real-time ubicomp applications. Using RCSM's CA-IDL, the context-awareness of a real-time object can be expressed by associating context expressions along with the methods declared in the object's interface. A real-time context-sensitive object is a type of object that associates a particular context along with at least one of its prioritized methods in the object interface. We define this type of methods as real-time context-sensitive methods. These methods are invoked whenever their corresponding objects are activated and their scheduling criteria are met. In Section 6.1, we discuss how a developer can use our CA-IDL to specify their interests in different contexts. We then discuss how to specify context-sensitive interfaces in Section 6.2.

6.1. Context Specification

6.1.1. Context Categories

Specification of contexts in an object interface is limited by the context data supported by the host device. The types of context available within a device, in turn, are limited to the

Table 1. Examples of context-category structure definitions in RCSM.

RCSMContext
DeviceSpecificContext
{double battery_power;
double light_intensity;
double net_transmission_rate;};
	RCSMContext
	EnvironmentSpecificContext
	{unsigned int number_peer_devices;
	char [16] location;};
		RCSMContext
		UserSpecificContext
		{unsigned int
		calendar_usage_rate;};

device's context-sensing capabilities. We perform the following steps (Whenever RCSM is ported to a new device):

a. Classify the contexts in the following categories:

   i. Device-specific context—context information that is specific to the device, such as the number of objects activated or currently running, remaining battery power, and current time.

   ii. Environment-specific context—context information that is specific to the surrounding environment, such as the current location, the number of devices in the vicinity, light intensity, and current temperature.

   iii. User-specific context—context information that is specific to the user, such as user information and the number of times a user runs a particular application.

b. Define a structure type for each category. We accomplish this by using the RCSMContext keyword of our CA-IDL. Each structure type includes one or more data types to store values of different contexts that are detectable by RCSM. We uniquely identify each structure type by using one of the following CA-IDL keywords: DeviceSpecificContext, EnvironmentSpecificContext, and UserSpecificContext.

Several examples, which show the definitions of three structure types, corresponding to all three categories, for a particular device, are given in Table 1. As shown in Table 1, this device supports six different types of contexts: battery power, light intensity around the device, and the network bandwidth are device-specific contexts; the number of peer devices in the vicinity and the location of the device are environment-specific contexts; and the frequency of the user's usage of the calendar software is the user-specific context.

6.1.2. Context Variables, Temporal Operators, and Context Expressions

DEFINITION 9 *Context variables are variables in CA-IDL to specify different values for a specific context.*

An application developer can use a context-variable to express his/her interest in specific values of a context. We declare a context-variable in CA-IDL using the RCSMContext_var keyword and then specifying the corresponding structure type and a particular value for the context-variable according to its data type. The general structure for defining a context variable is shown below:

RCSMContext_var ⟨⟨category type⟩⟩ ⟨⟨variable name⟩⟩ where ⟨⟨structure field⟩⟩ op ⟨⟨constant expression⟩⟩

For example, we can use the examples presented in Table 1 to specify context-variables C1, C2, and C3 as shown in (1), (2), and (3). As (1) and (2) show, C1 is a context-variable to store a specific value—‘‘GWC329’’—for the location context, and C2 stores the boolean value to indicate whether the number of peer devices is more than 1. As (3) shows, C3 is a context-variable to store the boolean-valued conjunction related to two different contexts: number of peer devices and network bandwidth. The bold letters in (1)–(3) indicate that they are CA-IDL-specific keywords.

RCSMContext_var DeviceSpecificContext C1 **where** (location = ‘‘GWC329’’) (1)

RCSMContext_var EnvironmentSpecificContext C2 **where** (number_peer_devices > 1) (2)

RCSMContext_var EnvironmentSpecificContext C3 **where** (number_peer_devices > 2) **and** (net_transmission_rate >= 40) (3)

It is important to note that the actual semantic interpretation of the context-variables is application-specific, and hence should be part of the object implementation rather than a feature of the core middleware services.

DEFINITION 10 *Temporal operators are operators to specify temporal relationships among multiple context variables in CA-IDL.*

Most of the operators as summarized in Table 2 are conceptually similar to the basic regular expression operators. A1 and A2 in Table 1 are two context variables.

Table 2. Temporal operators in CA-IDL for constructing context expressions.

Operator	Usage	Description
Union: ‘‘ + ’’	$[(A1 + A2)t]$	Either A1 or A2 is true for the last time period *t*
Concatenation: ‘‘ ∧ ’’	$[(A1 \wedge A2)t]$	Both A1 and A2 are true for the last time period *t*
Singular: ‘‘( )’’	$[(A1)t]$	A1 is has been true for the last time period *t*
Precedence: ‘‘- > ’’	$[(A1\text{-} > A2)t]$	A2 becomes true within *t* time units A1’s being true

DEFINITION 11 *Context expressions are expressions to represent the relations among context variables using temporal operators.*

We define a context expression using the RCSMContext_var keyword and then specifying the relations between two context variables or expressions. In this sense, we can define a complex context expression recursively by using simpler context expressions. For example, assuming we are interested in the condition that either C1 or C2 (as previously declared) is true for the last 10 seconds, we can use the union operator ‘‘ + ’’ in Table 2 to declare a context expression E1 as follows:

RCSMContext_var E1 where [(C1 + C2) 10] (4)

6.2. Context-Sensitive Interface Specification

An application developer can define an interface for a context-sensitive real-time object by associating context variables and context expressions in CA-IDL with the method signatures as follows:

1. Annotate a method of a context-sensitive interface with either an [incoming] or an [outgoing] tag.

2. Add an [activate-at-context *x*] tag where *x* represents a context-variable or context expression.

An [incoming] tag signifies that the corresponding method should be invoked only after data is available from a remote object following the creation of a context-triggered communication channel. An [outgoing] tag signifies that the method should be invoked first to generate the necessary data that will be transmitted to a remote method with an [incoming] tag. It is necessary for any pair of methods to be of opposite types to be able to communicate through a context-triggered communication channel. For example, we can define an interface—ContextSensitivePrinter—for a hypothetical context-sensitive object for facilitating printing services by dynamically discovering printers in a room, GWC 329. As shown in (5), the ContextSensitivePrinter interface has two methods: The first method—SendDocumentstoPrinter is invoked whenever C1 is true. From (1), we can see that C1 refers a particular location ‘‘GWC329’’. As such, whenever the device detects that it is in ‘‘GWC329’’, the SendDocumentstoPrinter method is invoked. The outgoing tag indicates that this method should generate the necessary data for use by the remote object, if in fact a context-triggered communication channel is established with another device (possibly a printer). The second method—NotifyUser is invoked to ask for user’s preference for a specific printer whenever more than one printer is detected in GWC329 for more than 5 seconds. For NotifyUser, we can use the context-variables C1 and C2 as declared in (1) and (2).

```
interface ContextSensitivePrinter {
    [outgoing][activate at C1]void SendDocumentstoPrinter(... ...);
    [outgoing][activate at (C1 ∧ C2)5] void NotifyUser (... ...);
};                                                                    (5)
```

7. Development of Adaptive Object Containers (ADC)

In this section, we discuss how ADCs can be used to perform Steps (R2), (R4), and (R8) in RCSM. An ADC, which is generated by the CA-IDL compiler based on a context-sensitive interface specification, has the following main tasks:

- Register the corresponding context-sensitive object and its interests in context with the R-ORB.
- Periodically receive the context data from the R-ORB.
- Analyze the data to check if the context for the associated context-sensitive object is true.
- Activate the associated context-sensitive object and invoke the method as indicated by the R-ORB.

We describe Steps (R2), (R4), and (R8) in Sections 7.1, 7.2, and 7.3, respectively.

7.1. Interface-Specific ADC Generation

The CA-IDL compiler generates an ADC for a specific context-sensitive interface by generating the following objects as shown in Figure 9.

RTDispatcher: RTDispatcher is responsible for activating the object and its methods through the RTObjectBase. If multiple methods need to be invoked RTDispatcher uses the priority information of a method to decide the method invocation. RTDispatcher invokes a method only after receiving an event from the R-ORB. This event, which we will define in Section 8, indicates that R-ORB has found a compatible object in a remote device and has established a context-triggered communication channel.

RTObjectBase: This object has the definitions of the methods described in the interface specified in CA-IDL. An application developer extends this object to develop the implementations of the context-sensitive object.

RTObjectContextReflector: This object (RTOCR) is responsible for detecting the suitability of method invocation in the current context. Based on the context-variables or expressions specified for each method, the CA-IDL compiler generates an RTOCR. In

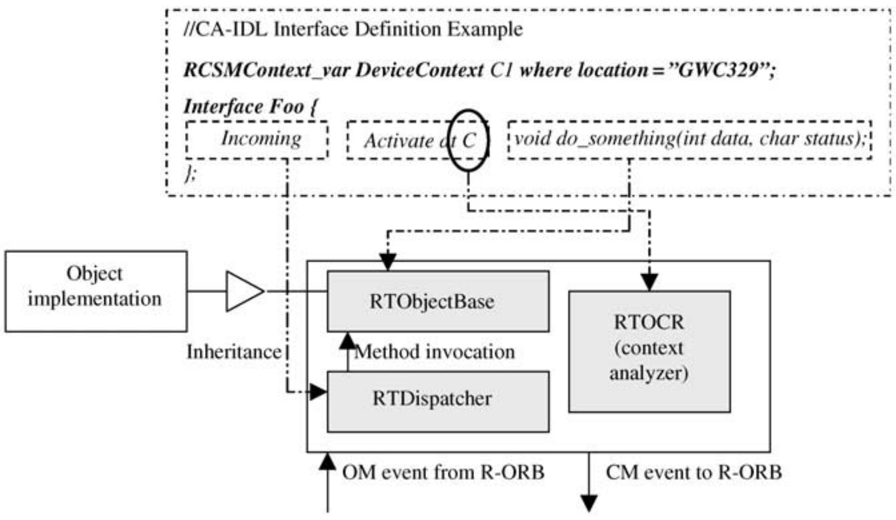

Figure 9. The architecture of an ADC.

order to facilitate Step (R2) the compiler generates two tables in each RTOCR to facilitate this task. The first table is called the CV-Table, which stores the information related to context variables in an interface. The second table is the CE-table, which stores the information related to the context expressions. We illustrate each table as Tables 3 and 4 and explain their columns in Tables 5 and 6, respectively. An RTOCR also has a CB-table to store the latest values of the contexts received from the underlying R-CAP framework.

Table 3. A CV-table in an ADC.

Row	Context variable	Operator	Constant value	*V*	True for duration	Method ID
1	...	...	...	...	...	Method 1
2	...	...	...	...	...	Method 2
n	...	...	...	...	...	Method *n*

Table 4. A CE-table in an ADC.

Row	Context left entry	Operator	Context right entry	Specified duration	*V*	True for duration	Method ID
1	...	...	...	...	...	...	Method 1
2	...	...	...	...	...	...	Method 2
n	...	...	...	...	...	...	Method *n*

Table 5. Explanation of the fields in a CV-table.

Context Variable	Stores the identifier for a particular context variable.
Operator	Stores the identifier for the operator (boolean) that is used to specify a specific value for the context variable.
Constant Value	Stores the constant value of a context variable.
V	Stores the True/False to indicate whether the value stored in the Constant Value field is true in the current context.
True for Duration	Stores the elapsed time since when the *V* field is True, if it is true.
Method ID	Stores the numerical identifier of the corresponding method.

Table 6. Explanation of the fields in a CE-table.

Context Left Entry	Stores the left portion of a context expression.
Operator	Store the identifier for the operator (temporal) that is used to build the corresponding context expression.
Context Right Entry	Stores the right portion of a context expression.
Specified Duration	Stores the time value for which the corresponding context expression should be true.
V	Stores the True/False to indicate whether the corresponding context expression is true in the current context.
True for Duration	Stores the elapsed time since when the *V* field is True, if it is true.
Method ID	Stores the numerical identifier of the corresponding method.

7.2. Runtime Context Analysis

An RTOCR in an ADC uses the CV-table, the CE-table, and CB-table to perform Step (R2) as follows:

a. For each row in the CV-table, the RTOCR compares the corresponding current value in the CB-table with the value stored in the Context Variable column. If the value is true, it sets the V column of the corresponding row to *T*. If the *V* field has been true before, it updates the ‘‘True for Duration’’ field by adding the time elapsed since last time it had performed this step. If *V* is *T*, it notifies the R-ORB that the corresponding object is suitable for activation in the current context. The notification includes the Method ID and the corresponding interface name.

b. For each row in the CE-table, RTOCR performs the same operations as in Step (a), except that it evaluates if the entire context expression is true for the Specified Duration using the current values from the CB-table.

7.3. Context-Triggered Object Activation and Method Invocation

Although an ADC is responsible for activating a context-sensitive object, an ADC and the R-ORB work cooperatively to identify the suitability of context-triggered object activation as follows:

a. Following a notification from RTOCR in Steps (a) and (b), an R-ORB initiates an object discovery procedure to discover remote objects for the local method.

b. If a compatible object in a remote device is found, the R-ORB notifies the RTDispatcher object of the ADC. This notification corresponds to Step (R7).

c. The RTDispatcher object checks the scheduling criteria to determine if the method can be invoked. If the criteria are met, the RTDispatcher activates the object and invokes the method as shown in Figure 9. This activation corresponds to Step (R8).

8. Context-Sensitive Communications Using R-ORB

In this section, we will discuss how R-ORB performs Steps (R5) and (R6) to provide the context-sensitive communication facilities in RCSM. R-ORB itself behaves as a context-sensitive object to decide when to establish context-triggered communication channel with a remote R-ORB so that a local object can exchange data with a remote object. We will describe the context-sensitive behavior of R-ORB in Section 8.1, and describe Steps (R5) and (R6) in Section 8.2.

8.1. R-ORB's Context-Sensitive Actions

DEFINITION 12 *A context-match event is a signal generated by an ADC to notify the R-ORB that a context variable or context-expression in an interface is currently true.*

DEFINITION 13 *An object-match event is a signal generated by an R-ORB to notify an ADC that a remote object is found that is suitable to communicate with the ADC's application object.*

In Definition 13, by suitability, we mean matching of the object interface signatures, including number of parameters and parameter types, etc. The ADC's notification to R-ORB in Step (a) in Section 7.2 is the context-match (CM) event. The R-ORB's notification to the ADC in Step (d) in Section 7.3 is the object-match (OM) event. Using these signals the ADCs and the R-ORB in a device to coordinate their actions. We now define the necessary and sufficient conditions for establishing a context-triggered communication channel:

Table 7. Context variables of an R-ORB.

Context variables	Description	Type
X_n	Number of new devices detected in the vicinity	Natural number
X_l	Number of existing devices that are no longer detected	Natural number
E_{CS}	Whether any CM event is pending with respect to a context-sensitive object	Boolean
E_S	Whether any OM event is pending with respect to a context-sensitive object	Boolean

Necessary condition:

$$\exists \delta_i | \text{pending}(O_m, \text{CM}) = \text{True} \wedge O_m \in \text{methods}(\delta_i) \tag{6}$$

where δ_i represents a context-sensitive object in the local device ‘‘pending(O_m, CM)’’ represents that a CM event corresponding to method O_m has occurred, and ‘‘methods(δ_i)’’ represents the set of methods of a context-sensitive object δ_i.

Sufficient condition:

$$\exists \delta_i | \text{pending}(O_m, \text{OM}) = \text{True} \wedge O_m \in \text{methods}(\delta_i) \tag{7}$$

where ‘‘pending$(\delta_i O_m, IM)$’’ indicates that a recent OM event corresponding to method O_m has occurred.

The condition in (6) is necessary with respect to that in (7), because if the CM event is false in (6), the OM event is false by default, because the R-ORB does not initiate an object discovery procedure, which is executed to decide if (7) holds true. The condition in (7) is sufficient with respect to that in (6) since (7) is being true indicates the success of an object discovery procedure, which is executed only after the condition (6) becomes true.

The occurrence of a CM event implies that at least one object in the local device is ready to be activated in the current context, and hence R-ORB should take the necessary actions to find a communication partner for this object. Thus, we can see that it is beneficial to consider R-ORB as a context-sensitive object itself. As such, we define the context variables for R-ORB in Table 7.

Table 7 shows that R-ORB essentially is responsible for ‘‘sensing’’ the following contexts: both CM and OM events and the devices that are currently in the vicinity. With the information in Table 7, we now define the context-sensitive operations of R-ORB as shown in Table 8.

In Table 8, the values of K_{PD} and K_{DL}, can be set to any natural number and thus can be used as threshold values to manage the ‘‘level of sensitivity’’ of the R-ORB. Similarly,

Table 8. Context-sensitive operations of an R-ORB.

	Context expressions	Action taken by R-ORB if context is true
1.	$[\{(E_{CS} = \text{True}) \wedge (X_n > K_{PD})\}, t_A]$	Initiate communication with the peer devices to discover the objects in these devices
2.	$\{(X_l \geq K_{DL}), t_B\}$	Update the remote object information cache
3.	$\{(E_S = \text{True}), t_C\}$	Establish a context-triggered communication channel with a device and notify the ADC to activate the object in the local device

the parameters—t_A, t_B, and t_C can be set to any real number and changed dynamically. This implies that degree of context sensitivity of R-ORB can be adapted in response to variations in context data over a period.

8.2. Context-Sensitive Object Discovery and Context-Triggered Communication Channel Establishment

In order to facilitate Steps (R5) and (R6), R-ORB broadcasts ODD messages so that the remote R-ORBs in other devices can discover the objects in the local device. An ODD message has the following information:

$$\{\text{Device address, interface id, method id, }\{\text{parameter list, parameter type}\},\ \text{keywords}\} \tag{8}$$

where Device address corresponds the unique address of the device (e.g., IP address), object id corresponds to a unique identifier of an interface, and method id corresponds to the method that satisfies (6). Parameter list and parameter type represent the parameters and their types of the method. Keywords represent a list of optional keywords that can be used to match the services of two objects for exchanging data. To prevent unnecessary bandwidth consumption, R-ORB only broadcasts the information of the context-sensitive objects that already have pending context-match events, i.e., these objects satisfy the necessary condition for a connection establishment. Hence, the ODD broadcasting is completely turned off when no CM event is received from the ADCs. Steps (a) and (b) correspond to Step (R5), and the remaining steps correspond to Step (R6).

Whenever the R-ORB receives an ODD broadcast, it takes the following actions:

a. If there is any method satisfying the condition in (6), perform a matching by comparing the local method and its objects with the information contained in the ODD message.

b. If a matching is found in Step (b), generate an OM event for this method and continue to Step (c). Otherwise, stop.

c. If the method, for which the OM event is generated in Step (b), is of type outgoing, notify the corresponding ADC to invoke the method and retrieve the results. If the method's type is incoming, continue to Step (e).

d. Periodically check if the ADC that was notified in Step (c) has completed the method invocation and has passed the object data to the R-ORB. When the data is received from the ADC, continue to Step (f).

e. Periodically check for the data transmission from the R-ORB of the device from which the ODD message in Step (a) came from. When the data is received from the remote R-ORB, continue to Step (g).

f. Create a point-to-point communication channel with the remote R-ORB of the device from which the ODD message in Step (a) came from. Transmit the object data received at Step (d) to the remote R-ORB. Terminate the channel after the transmission is completed.

g. Notify the ADC of the method in Step (a) to invoke this method and pass in the data received in Step (e) to the ADC.

9. An Illustrative Example

In this example, we illustrate Steps (R1)–(R8). The requirements of our target system are as follows: The system is a network of embedded sensors. Two different types of sensors, motion and noise detectors, are used to monitor an environment. As shown in Figure 10, a mobile robot collects the data from these sensors whenever the robot is within the 10 m of either sensor. Both types of sensors are stationary and have limited radio transmission range of up to 10 m. We assume Object M, Object N, and Object MB provide the desired functionality in the motion detectors, noise detectors, and the mobile robot respectively.

Step (R1): In this step, we provide the context-sensitive interfaces of Object A, B, and C based on the procedure explained in Section 5.

<u>Object MB</u>

```
  //Name: Mobile Robot Object
  //Define a context variable
  RCSMContext_var NetworkContext C where (number_peer_devices > 0);
  //Interface Definition
  interface MB {
```

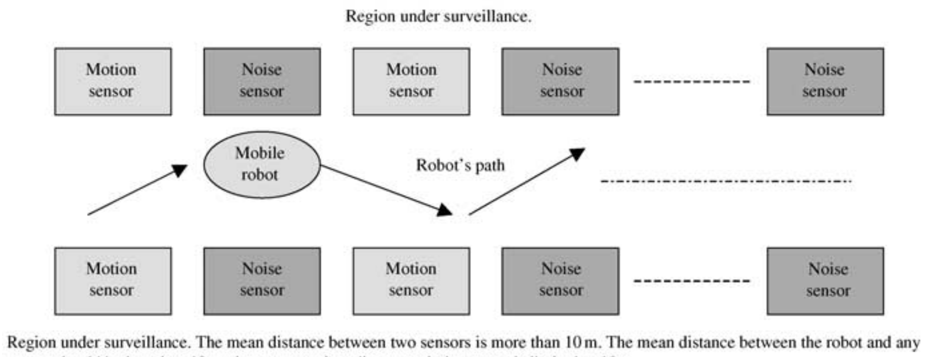
Region under surveillance.
Motion sensor
Noise sensor
Motion sensor
Noise sensor
Noise sensor
Mobile robot
Robot's path
Motion sensor
Noise sensor
Motion sensor
Noise sensor
Noise sensor
Region under surveillance. The mean distance between two sensors is more than 10 m. The mean distance between the robot and any sensor should be less than 10 m since a sensor's radio transmission range is limited to 10 m.

Figure 10. A mobile robot uses RCSM's context-sensitive communications to collect surveillance data whenever the robot is within the 10 m of a motion or a noise sensor.

```
    [incoming] [activate at C] receive_noise_data([in] string data);
    [incoming] [activate at C] receive_motion_data([in] string data);
  };
```

<u>Object M</u>

```
  //Name: Motion Data Collector
  //Define a context variable
  RCSMContext_var NetworkContext C where (number_peer_devices > 0);
  //Interface Definition
  interface M {
    [outgoing] [activate at C] exchange_motion_data ([out] string data);
  };
```

<u>Object N</u>

```
  //Name: Noise Data Collector
  RCSMContext_var NetworkContext C where (number_peer_devices > 0);
  //Interface Definition:
  interface N {
    [outgoing] [activate at C] exchange_noise_data ([out] string data);
  };
```

Tables 9–11 show the compatible object-method signatures for each object to facilitate context-sensitive communications.

Step (R2): Based on these interfaces, in this step the CA-IDL compiler generates three ADCs based on the procedure explained in Section 6. The context-variable tables of these ADCs are shown in Tables 12–14.

Table 9. Compatible interface-method signatures for Object MB in the mobile robot.

Local Object-Method Signatures	Compatible Object-Method Signatures
Object MB: receive_noise_data([in] string data)	Object N: exchange_noise_data ([out] string data);
Object MB: receive_motion_data([in] string data);	Object M: exchange_motion_data ([out] string data);

Table 10. Compatible interface-method signatures for Object M in the motion detectors.

Local Object-Method Signatures	Compatible Object-Method Signatures
Object M: exchange_motion_data ([out] string data);	Object MB: receive_motion_data([in] string data);

Table 11. Compatible interface-method signatures for Object N in the noise detectors.

Local Object-Method Signatures	Compatible Object-Method Signatures
Object N: exchange_noise_data ([out] string data);	Object MB: receive_noise_data([in] string data)

Table 12. Context-variable table for Object MB in the mobile robot.

Row	Context variable	Operator	Constant expression	Specified duration	*V*	True for duration	Method Id
1	Number_peer_devices	>	1	—	—	—	1
2	Number_peer_devices	>	1	—	—	—	2

Table 13. Context-variable table for Object M in the motion detectors.

Row	Context variable	Operator	Constant expression	Specified duration	*V*	True for duration	Method Id
1	Number_peer_devices	>	1	—	—	—	1

Table 14. Context-variable table for Object N in the noise detectors.

Row	Context variable	Operator	Constant expression	Specified duration	*V*	True for duration	Method Id
1	Noise_level	>	0	—	—	—	1

Step (R3): Since each sensor is more than 10 m apart from the other sensors, it cannot detect other sensors. If the mobile robot is not within the 10 m, the R-CAP periodically propagates the number of peer devices (which is 0) to the ADCs. Whenever the robot is in the vicinity, the number becomes 1. The same is true for the mobile robot as well. Whenever it is within 10 m of some sensors, the number becomes 1. Otherwise, it stays 0.

Assume, the robot is within the range of a motion detector. The following sequences of actions occur:

Step (R4): According to Tables 9 and 10, both Object MB and Object M satisfy the condition expressed in (6), because for both the robot and the sensor the number_peer_devices is 1. As such, the ADCs in both devices generate CM events.

Step (R5): Upon receiving the CM event from the ADC, the R-ORB in the robot broadcasts the following ODD messages:

$$\{192.168.0.12,\ \text{MB},\ \text{receive\_noise\_data},\ \{\text{data}, \text{string}\},\ \text{none}\} \tag{9}$$

$$\{192.168.0.12,\ \text{MB},\ \text{receive\_motion\_data},\ \{\text{data}, \text{string}\},\ \text{none}\} \tag{10}$$

Here, we assume the robot's IP address is 192.168.0.12. Note that the robot broadcasts two ODD messages—one for each method in the Object MB, since both methods can be activated in the current context according to Object MB's interface specification. The R-ORB in the motion detector also broadcasts an ODD message as follows:

$$\{192.168.0.14,\ \text{M},\ \text{exchange\_motion\_data}, \{\text{data},\ \text{string}\},\ \text{none}\} \tag{11}$$

Here, we assume the motion detector's IP address is 192.168.0.14.

Steps (R6)–(R8) in the mobile robot: After the receiving the ODD broadcast message in (11) from the motion detector, the R-ORB in the mobile robot compares the information in the ODD message with the information in Table 9. Since according to the table, the receive_noise_data is compatible with an exchange_motion_data method, R-ORB generates an OM event to indicate that a compatible object is found. Since the receive_noise_data is the local method and is of type incoming, R-ORB in the robot follows Steps (a), (b), (c), (e), and (g) in Section 8.2 to receive the motion data for the Object MB.

Steps (R6)–(R8) in the motion detector: After the receiving the ODD broadcast messages in (9) and (10) from the mobile robot, the R-ORB in the motion detector compares the information in these ODD messages with the information in Table 10. Since according to the table, the exchange_motion_data is only compatible with a receive_motion_data method, R-ORB stops in Step (b) of Section 8.2 when it analyzes the message in (9). However, when R-ORB analyzes the information in the message in (10), it generates an OM event to indicate that a compatible object is found. Since the exchange_motion_data is the local method and is of type outgoing, R-ORB in the motion detectors follows Steps (a)–(f) in Section 8.2 to send the motion data to the mobile robot.

When the robot is near a noise sensor, similar procedure is followed except Table 11 is used instead of Table 9.

10. Discussion

In this paper, we have presented RCSM, an adaptive and object-based middleware to facilitate context-sensitive communications among real-time applications in ubicomp environments. RCSM's OMA-compliant architecture allows application software developers to interoperate RCSM with other industry-standard middleware technologies. We also presented RCSM's support for development- and run-time facilities for real-time ubicomp applications. Specifically, we discussed RCSM's context-sensitive object-based development process using its CA-IDL and the mechanism for providing application-specific context-processing support using ADCs. RCSM's R-CAP framework provides the low-level context acquisition and propagation services to isolate the difficult task of context collection from context-awareness. We also presented R-ORB, which is the communication backbone in RCSM to provide both client–server and context-sensitive communications in *ad hoc* networks. The effect of high-frequency changes in the contexts on the deadlines of real-time objects needs to be explored further to provide context-sensitive real-time scheduling support in RCSM. Additional work needs to be done to make R-ORB more energy efficient for *ad hoc* object communication in constantly mobile environments. It is also our plan to develop a bridge between the R-ORB and the CORBA-IIOP to enable seamless interoperability with RT-CORBA objects.

We are evaluating different features of RCSM in an ubicomp test bed, called Smart Classroom, which provides a middleware infrastructure for university classrooms to facilitate collaborative learning among students. The test bed consists of various types of PDAs, short-range radio transceivers, and sensors connected through RCSM over an integrated *ad hoc* network and fixed network environments.

Acknowledgments

This research is supported in part by National Science Foundation under grant number ANI-0123980. We would like to thank the reviewers for their valuable suggestions in improving the quality and readability of this paper. We also would like to thank Richard Wang for implementing the CA-IDL compiler.

References

Abowd, G., and Mynatt, E. D. 2000. Charting past, present, and future research in Ubicomp. *ACM Transactions on Computer Human Interaction* 7(1): 29–58.

Aura Project. 2000. Carnegie Mellon University. Project location: http://www-2.cs.cmu.edu/ ~ aura/.

Bennett, F., Clarke, D., Evans, J. B., Hopper, A., Jones, A., and Leask, D. 1997. Piconet—Embedded Mobile Networking. *IEEE Personal Communications* 4(5): 8–15 (paper location: http://www.uk.research.att.com/pen/).

Bernstein, P. A. 1996. Middleware: A model for distributed system services. *Communications of the ACM* 39(2): 86–98.

Bluedrekar Project. 2000. IBM Research. Project location: http://www.research.ibm.com/BlueDrekar/.

Bluetooth. 1999. *Specification of the Bluetooth System*. Core, version 1.0 B. Specification location: http://www.bluetooth.com/.

Broch, J., Maltz, D. A., Johnson, D. B., Hu, Y.-C., and Jetcheva, J. 1998. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the 4th International Conference Mobile Computing and Networking (MOBICOM 98)*, Dallas, USA, pp. 85–97.

Campbell, A. T., Kounavis, M. E., Liao, R. R.-F., and Angin, O. 1998. The mobiware toolkit: programmable support for adaptive mobile networking. *IEEE Personal Communications* (paper location: http://comet.ctr.columbia.edu/mobiware/).

Chen, G., and Kotz, D. 2001. A survey of context-aware mobile computing research. *ACM Operating Systems Review* 35(1) (paper location: http://www.cs.dartmouth.edu/reports/authors/Kotz,David.html).

Chen, K., Shah, S. H., and Nahrstedt, K. 2002. Cross-layer design for data accessibility in mobile ad hoc networks. *Journal of Wireless Personal Communications* 21(2): 49–76.

DARPA (Defense Advanced Research Project Agency). Ubicomp. Location: http://www.darpa.mil/ito/research/uc/index.html.

Dey, A. K. 2001. Understanding and using context. *Personal and Ubicomp, special issue on Situated Interaction and Ubicomp* 5(1) (paper location: http://www.cc.gatech.edu/fce/publications.html).

Gokhale, A., and Schmidt, D. 1999. Techniques for optimizing CORBA middleware for distributed embedded systems. In *Proceedings of the IEEE INFOCOM 99*. New York, NY (paper location: http://www.cs.wustl.edu/ ~ schmidt/corba-research-realtime.html).

Haahr, M., Cunningham, R., and Cahill, V. 1999. Supporting CORBA applications in a mobile environment. In *Proceedings of the Fifth ACM/IEEE International Conference Mobile Computing and Networking (MobiCom 99)*, pp. 36–47.

HomeRF Working Group. 2000. *HomeRF Protocol Standards*. Specification location: http://www.homerf.org/learning_center/.

IBM. 2000. What is Pervasive Computing? Location: http://www-3.ibm.com/pvc/pervasive.shtml.

IEEE (The Institute of Electrical and Electronics Engineers). 1997. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE 802.11*. Specification location: http://standards.ieee.org/getieee802/.

IEEE (The Institute of Electrical and Electronics Engineers). 2001. *Working Group for Wireless Personal Area Networks*. Group location: http://grouper.ieee.org/groups/802/15/.

IETF (Internet Engineering Task Force). 2001. *Mobile Ad Hoc Networks Charter*, http://www.ietf.org/html.charters/manet-charter.html.

IrDA (The Infrared Data Association). 2000. *IrDA Protocol Standards*. Specification location: http://www.irda.org/standards/standards.asp.

Kon, F., Roman, M., Liu, P., Mao, J., Yamane, T., Magalhaes, L. C., and Campbell, R. H. 2000. Monitoring, security, and dynamic configuration with the dynamicTAO reflective ORB. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms and Open Dist. Processing (Middleware 2000)*, New York, NY (paper location: http://devius.cs.uiuc.edu/2k/dynamicTAO/).

Marmasse, N., and Schmandt, C. 2000. Location-aware information delivery with comMotion, In *Proceedings of the 2nd International Symposium on Handheld and Ubicomp (HUC 2K), Lecture Notes in Computer Science*, 1927 (paper location: http://www.media.mit.edu/ ~ nmarmas/research.html).

Mascolo, C., Capra, L., Zachariadis, S., and Emmerich, W. 2002. XMIDDLE: A Data-Sharing Middleware for Mobile Computing. *International Journal on Wireless Personal Communications* 21(1): 77–103.

Murphy, A., Picco, G., and Roman, G.-C. 2001. LIME: A middleware for physical and logical mobility. In *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS 2001)*, Phoenix, USA (paper location: http://www.cs.rochester.edu/u/www/u/murphy/papers/index.html).

Nahrstedt, K., Xu, D., Wichadakul D., and Li, B. 2001. QoS-Aware Middleware for Ubiquitous and Heterogeneous Environments. *IEEE Communications* (paper location: http://choices.cs.uiuc.edu/gaia/html/publications.htm).

OMG (Object Management Group). 2000. *CORBA 2.5 Specification*. Specification location: http://www.omg.org/cgi-bin/doc?formal/01-09-01.

Oxygen Project. 2000. MIT Laboratory for Computer Science. Project location: http://www.oxygen.lcs.mit.edu/.

Rabaey, J. M., Ammer, M. J., Silva, M. Josie, Patel, D., and Roundy, S. 2000. PicoRadio Supports Ad Hoc Ultra-Low Power Wireless Networking. *Computer* (paper location: http://bwrc.eecs.berkeley.edu/Research/Pico_Radio/Default.htm).

Rappaport, T. S. 1995. *Wireless Communications: Principles and Practice*. Prentice Hall, Inc.

Royer, E. M., and Toh, C.-K. 1999. A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks, *IEEE Personal Communications*, pp. 46–55. Paper location: http://www.cs.ucsb.edu/ ~ eroyer/publications.html.

Schreiber, R. 1995. Middleware Demystified. *Datamation* 41(6): 41–45.

TSpaces Project. 2000. *IBM Research*. Project location: http://www.almaden.ibm.com/cs/TSpaces/.

Vaidya, N. H. 2001. Mobile *Ad Hoc* Networks: Routing, MAC, and Transport Issues. *Tutorial presented in the seventh Annual Int'l Conf. Mobile Computing and Networking (Mobicom 2001)*. Tutorial location: http://www.crhc.uiuc.edu/ ~ nhv/presentations.html.

W3C (World Wide Web Consortium). 2001. *W3C Architecture Domain: Extensible Markup Language*. Location: http://www.w3.org/XML/.

Weiser, M. 1991. The computer for the twenty-first century. *Scientific American* 265(3): 94–100.

Weiser, M. 1993. Some computer science issues in Ubicomp. *Communications of the ACM* 36(7): 75–84.

Yau, S. S., and Karim, F. 2001a. Reconfigurable context-sensitive middleware for ADS applications in mobile ad hoc network environments. In *Proceedings of the 5th IEEE International Symposium on Autonomous Decentralized Systems (ISADS 2001)*. Dallas, USA, pp. 319–326. Paper location: http://www.eas.asu.edu/ ~ rcsm.

Yau, S. S., and Karim, F. 2001b. Context-sensitive distributed software development for Ubicomp environments. In *Proceedings of the 25th International Computer Software and Applications Conference (COMPSAC 2001)*. Chicago, USA, pp. 263–268. Paper location: http://www.eas.asu.edu/ ~ rcsm.

Yau, S. S., Karim, F., Wang, Y., Wang, B., and Gupta, S. 2002. Reconfigurable context-sensitive middleware for pervasive computing. *IEEE Pervasive Computing* 1(3): 33–40.

Zimmerman, T. 1996. Personal area networks. *IBM Systems Journal* 35(3): 609–617.

Dr. Stephen S. Yau is currently professor of the Computer Science and Engineering Department at Arizona State University. He served as the chair of the department in 1994–2001. He was previously on the faculty of the University of Florida, Gainesville and Northwestern University, Evanston, Illinois. His current research is in the areas of middleware for ubiquitous computing, distributed computing, and software engineering.

He is a Fellow of the Institute of Electrical and Electronics Engineers (IEEE) and a Fellow of the American Association for the Advancement of Science. He received numerous awards, including the Louis E. Levy Medal of the Franklin Institute, the Tsutomu Kanai Award and the Richard E. Merwin Award of the IEEE Computer Society, the IEEE Centennial Medal, the IEEE Third Millennium Medal, the International Federation for Information Processing (IFIP) Silver Core Award, and the American Federation of Information-Processing Societies (AFIPS) Special Award.

He served as the president of the IEEE Computer Society and the American Federation of Information Processing Societies, and the editor-in-chief of Computer magazine. He has also served as a member of the Board of Directors of the Computing Research Association.

Fariaz Karim received a Ph.D. and a BS in Computer Science from Arizona State University in 2003 and 1997 respectively. His research interests include ubiquitous computing, distributed object computing middleware, software-hardware co-design of embedded systems, mobile computing, and component-based software development. He is currently a senior engineer at Intel Corporation, Oregon. He is a member of ACM, IEEE, and UPE.