Reconfigurable Context-Sensitive Middleware for Pervasive Computing

Context-sensitive applications need data from sensors, devices, and user actions, and might need ad hoc communication support to dynamically discover new devices and engage in spontaneous information exchange. Reconfigurable Context-Sensitive Middleware facilitates the development and runtime operations of context-sensitive pervasive computing software.

Stephen S. Yau, Fariaz Karim, Yu Wang, Bin Wang, and Sandeep K.S. Gupta
Arizona State University

A principal goal of pervasive computing is to make the actual computing part of it and its enabling technologies essentially transparent.[1–3] This transparency is partially possible because a pervasive computing environment is a collection of embedded, wearable, and handheld devices wirelessly connected, possibly to fixed network infrastructures such as the Internet. Devices in such environments have serious resource constraints and high vulnerability, and they form numerous webs of short-range and low-power mobile ad hoc networks to exchange information. This networking characteristic distinguishes pervasive computing environments from existing wireless and mobile phone networks, where fixed infrastructures enable communication.

Existing system prototypes and literature indicate that pervasive computing applications operating in ad hoc network environments usually exhibit two characteristics: context sensitivity and ad hoc communication.[3–7] An application software system's context is any detectable and relevant attribute of the software's host device, the software's user, the host device's surrounding environment, and the interaction between the host device and other devices.[8–10] Context sensitivity (or context awareness) is an application software system's ability to sense and analyze context from various sources; it lets application software take different actions adaptively in different contexts. Ad hoc communication describes the links among application software in various devices because they tend to be spontaneously established and terminated due to changing contexts, device mobility, and resource fluctuation.

Context sensitivity and ad hoc communication cut across system and application layers. Sometimes, applications need to exploit their context sensitivity to communicate with other devices.[11] A middleware-oriented approach can effectively address such issues by providing development and runtime support and by striking a balance between awareness and transparency to the application software.[12,13] In this article, we present Reconfigurable Context-Sensitive Middleware to show how to achieve this balance (see www.eas.asu.edu/~rcsm).[10,14]

What is RCSM?

We can divide currently available middleware for pervasive computing applications into two main categories based on how that middleware supports interaction among devices. The first category lets application software indirectly communicate with each other by writing to and reading from one or more shared (but structured) spaces.[15,16] Such middleware only treats contexts based on the data stored in tuple spaces, thereby ignoring the device's state (where the application software executes), network layers, and the surrounding environment as part of the overall context. The second category provides message-oriented semantics where application software objects see each other through mechanisms similar to remote procedure calls.[17–19]

Our RCSM (see the "Glossary" for this and other

1536-1268/02/$17.00 © 2002 IEEE

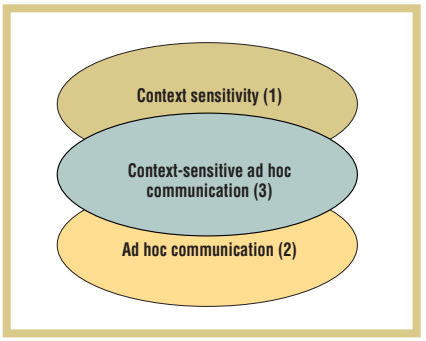

Figure 1. Pervasive computing applications with properties one, two, and three facilitated by our Reconfigurable Context-Sensitive Middleware (RCSM).

terms) is a middleware designed to facilitate applications that require context-awareness or spontaneous and ad hoc communication. However, the mechanisms in RCSM that provide these two properties are not completely independent. In other words, we support these two characteristics in RCSM in such a way that it becomes possible to facilitate a third type of applications that exhibits more complex behavior than context-aware applications or ad hoc collaborative applications do individually.

As Figure 1 shows, we characterize the property of this type of application as *context-sensitive ad hoc communication*. We consider all three types of application to be equally important for pervasive computing environments. Although finding examples of simply context-aware or simply ad hoc collaborative applications is easy, we have observed that context-aware ad hoc collaborative applications, such as context-aware messaging applications, are steadily emerging.[11,20,21]

RCSM also has several major features that address important middleware characteristics (see the related sidebar).

Glossary

ADC	Adaptive object container
CA-IDL	Context-aware interface definition language
CAEG	Context-aware ephemeral group
CTC	Context-triggered communication channel
FPGA	Field programmable gate array
RCSM	Reconfigurable Context-Sensitive Middleware
R-ORB	RCSM object request broker
R-GIOP	RCSM general inter-ORB protocol

Object-based development framework

Similar to mature middleware standards and prototypes such as CORBA, COM (Component Object Model), and TAO (the ACE ORB, Advanced Computing Environment Object Request Broker)[22] for fixed networks, RCSM provides an object-based framework for supporting context-sensitive applications. Taking an object-based approach in RCSM presents additional leverage beyond the benefits that simple object orientation provides. RCSM models context-sensitive application software as *context-sensitive objects*, which consist of two parts: a context-sensitive interface and a context-independent implementation. The interface encapsulates the description of the application's context awareness, whereas the implementation remains context free.

Application-specific adaptive object containers

For context-sensitive application software, RCSM provides adaptive object containers (ADCs) for runtime context data acquisition, monitoring, and detection. Each application object needs different contexts. Thus, as Figure 2 shows, RCSM's context-aware interface description language (CA-IDL) compiler generates a custom-made ADC tailored for a particular context-sensitive object. During runtime, the ADC communicates with the underlying system to acquire context data and then performs periodic context analysis as specified in the context-sensitive interface. It also communicates with the object implementation to activate different actions whenever the ADC detects suitable contexts as a result of the context analysis.

Context-sensitive object request broker

For pervasive application software that needs ad hoc communication support, we provide RCSM a context-sensitive object request broker (R-ORB) as the key mechanism for providing communication transparency for context-sensitive application software. R-ORB hides the intricacies of ad hoc networking. It also performs device and service discovery on the behalf of the context-sensitive objects whenever the objects' context specifications become true. We use a symmetric communication model in R-ORB to allow ad hoc and application-transparent information exchange between a pair of remote context-sensitive objects.

A classroom example: Two scenarios

Consider an instructor and his students using their PDAs to collaborate in a classroom. In one scenario, when the instructor is near the projection screen and the light in the classroom is turned down, the context indicates that the instructor is about to present some teaching material, which triggers the instructor's PDA to distribute the presentation material to the students' PDAs. In another scenario, students are divided into small groups to collaboratively solve a specific in-class problem. During group discussion, the instructor moves from one group to another to check the students' progress. When the instructor is near a particular group of students and he is facing the group, the instructor's PDA interprets this activity as an interest in that group. This context data lets the instructor's PDA join the student group for a short duration to download that group's discussion material.

These two scenarios show the following three common characteristics of pervasive computing applications.

Context sampling and detection

The facts that the instructor is near the projection screen and that the classroom becomes dark are two key pieces of data that the application software in the first

Middleware Characteristics

Middleware addresses two broad characteristics of pervasive computing: the trade-off between awareness and transparency and cooperation between development support and runtime services. However, it must address specific characteristics as well.

Broad characteristics

Looking into other research,[1–5] we learn an important lesson about building effective middleware for pervasive computing environments: how to provide a balance between awareness and transparency.[6] This contrasts with middleware for traditional computing environments, where providing complete transparency of the underlying technology and the surrounding environments is the ultimate goal. Such an approach does not work for pervasive computing applications because being aware of the surrounding environment is the key to their effectiveness. However, emerging applications for pervasive computing indicate that an appropriate level of transparency is a desirable feature to reduce the application software's complexity and to optimize the use of system resources in adverse conditions.

Although we mostly see the real benefits of middleware during software execution, the difference between just good and widely successful middleware is how easily it lets the application software developers exploit its various capabilities. Well-defined development processes and programming environments are equally important to developers.

Specific characteristics

In light of these broad characteristics, middleware for pervasive computing should also demonstrate specific characteristics to facilitate pervasive computing applications.

Uniform development support. Almost all the commonly used programming languages that exist today do not have basic support for expressing context awareness. Even if context-aware languages exist in the future, support for expressing context awareness on a conceptual level will most likely differ across different languages. This poses a problem when developers of context-sensitive application software need to reuse their designs in a different language, hardware, or operating system. As such, middleware must provide a uniform and common way to express the software's context awareness without restricting itself to a specific language, operating system, or environment.

Application-specific context acquisition, analysis, and detection. Providing a uniform and platform-independent interface for applications to express their need for different context data without knowing how that data is acquired is beneficial. Application software is often concerned with observing multiple contexts that follow a specific pattern of occurrence. Achieving this requires continuous data acquisition, analysis, and pattern detection. Middleware can help developers focus mainly on developing the applications' functionality rather than diverting their effort to hardware-specific issues.

Context-triggered action. Application software often decides what action to take based on the current context. The action could involve adapting to the new environment, notifying the user, communicating with another device to exchange information, or performing any other task. Middleware should provide the facilities for application software to define such context-triggered actions so as to transparently invoke them whenever the corresponding contexts are valid.

Transparent support for ad hoc communication. We need middleware to abstract the details of ad hoc communication from applications to facilitate interoperability independent of network type. The topologies in ad hoc networks change dynamically, and devices might not know each other a priori. On the other hand, a device in an ad hoc network could connect to a previously known computer (via a file or a Web server) in a wired network. Thus, middleware should transparently facilitate a task-oriented or publish-subscribe communication model so that application software can flexibly interact in different network environments. This middleware should also proactively discover new devices and functionalities, establish new communication links, and notify the application layer whenever it finds a compatible device. Furthermore, to let multiple devices transparently join and communicate in short-lived groups, the middleware should proactively check whether it should create new groups or continue to maintain existing ones, depending on application-specific criteria.

REFERENCES

1. A. Murphy, G. Picco, and G.-C. Roman, "LIME: A Middleware for Physical and Logical Mobility," *Proc. 21st Int'l Conf. Distributed Computing Systems,* IEEE CS Press, Los Alamitos, Calif., 2001, pp. 524–533.
2. C. Mascolo et al., "XMIDDLE: A Data-Sharing Middleware for Mobile Computing," *J. Wireless Personal Comm.*, vol. 21, no. 1, Apr. 2002, pp. 77–103.
3. T. Kindberg and A. Fox, "System Software for Ubiquitous Computing," *IEEE Pervasive Computing*, vol. 1, no. 1, Jan.–Mar. 2002, pp. 70–81.
4. D. Garlan et al., "Project Aura: Toward Distraction-Free Pervasive Computing," *IEEE Pervasive Computing*, vol. 1, no. 2, Apr.–June 2002, pp. 22–31.
5. B. Schilit, N. Adams, and R. Want, "Context-Aware Computing Applications," *Proc. IEEE Workshop Mobile Computing Systems and Applications*, IEEE CS Press, Los Alamitos, Calif., 1994, pp. 85–90.
6. L. Capra, W. Emmerich, and C. Mascolo, "Middleware for Mobile Computing: Awareness vs. Transparency," *Proc. 8th Workshop Hot Topics in Operating Systems,* IEEE CS Press, Los Alamitos, Calif., 2001, pp. 164-169.

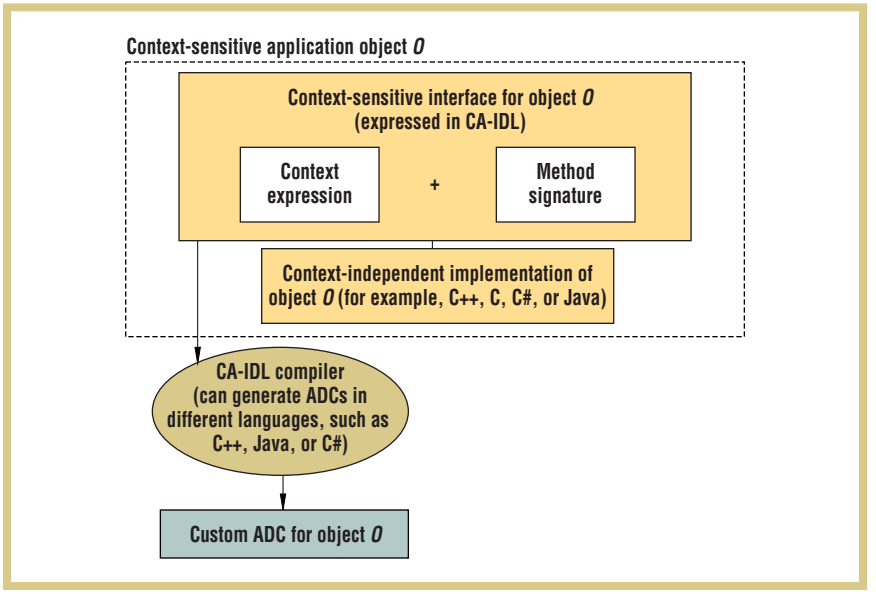

Figure 2. RCSM's context-aware interface description language (CA-IDL) compilers build application-specific adaptive object containers (ADCs) based on a context-sensitive interface description.

scenario needs. The second scenario requires the analysis of two context attributes: the instructor's location and direction.

Spontaneous context-aware ad hoc communication

The first scenario indicates that the instructor's PDA distributes the lecture materials to the students' PDAs as soon as appropriate contexts are valid. This also implies that the instructor's PDA must discover other PDAs in the vicinity, check that they do indeed "need" the lecture material, and establish temporary communication links.

Ad hoc ephemeral group establishment

The second scenario indicates that students' PDAs dynamically form a group, which is valid only for the class's duration. Moreover, the instructor's PDA dynamically joins a particular group of students for a short duration to download the group's solutions to the in-class problem.

RCSM's key capabilities

Although contexts can conceptually cover all aspects of a pervasive computing application's environment, the context captured by a specific application software is limited by the scope and capabilities of the host device's sensors and operating system.

Using context information is sometimes difficult due to complex sensor handling, context abstraction, and heterogeneous context sources.[23] How to incorporate context awareness into an application to make it more intelligent is also a practical problem. Moreover, because pervasive computing applications can have different requirements for using contexts, the current development process of such applications lacks a uniform and standard procedure for developers to follow.

Several research groups have tried to overcome the difficulties in developing context-sensitive applications. In the Context Toolkit, a predefined context is acquired and processed in context widgets and then reported to the application through application-initiated queries and callback functions.[23] The Tea project provides query primitives for applications to acquire context information.[24]

We chose to design our context-processing system to directly trigger the appropriate actions in an application object rather than have the object itself decide which method (or action) to activate based on context. Our primary motivation was to extend existing context-sensitive applications by adding new context sources. We also wanted to easily let multiple concurrent contexts trigger a specific action. As we mentioned earlier, we divide context-sensitive application software into two separate parts as shown in Figure 2. The first is an interface that encapsulates the application's context sensitivity. More specifically, this interface lists the contexts the application uses, a list of actions (functionality) the application provides, and a mapping between the specified contexts and these actions that clearly indicates when an action should be completed based on specific context values. The second part is the actual implementation of the actions that the application software must provide.

The important characteristic of this structure is that the implementation is completely isolated from context specification, meaning it is context independent. This is where RCSM's strength becomes visible. Using the context-sensitive interface, RCSM determines which context to monitor and which of the applications' actions to activate whenever a specified context is valid. The developer simply focuses on implementing the actions in his or her favorite language without worrying about context monitoring, detection, and analysis. Our CA-IDL lets application developers uniformly specify the context-sensitive object interface. Instead of being static context-processing mechanisms, the type of ADCs that can be generated by the CA-IDL compiler is virtually unlimited. For a different requirement, the application developer just needs to specify a different interface in the CA-IDL file and compile it to generate a new ADC. Figure 3 shows an example of a context-sensitive interface in CA-IDL. In this interface, three context variables are defined from our first scenario example: the instructor's being near the screen, darkness, and light. The expression $C1 \wedge (C2 \rightarrow C3)$ uses our operators to specify the temporal relationships among C1, C2, and C3. This particular expression indicates that C1 (instructor is near the screen) and C3 follows C2 (darkness follows light) must occur to transparently activate the *distribute* method.

If multiple context triggers occur for methods in the same object, the corresponding ADC invokes the corresponding methods in the order in which they appear in the object's interface. If these context triggers correspond to different objects, no conflict occurs because each object has its own ADCs that run independently of each other.

Spontaneous interaction support in R-ORB

Several challenges exist to letting application objects easily participate in spontaneous interaction. First, network connections in mobile ad hoc networks—the usual network technologies for enabling spontaneous networking—are often instantaneously established and terminated due to device mobility. Second, changes in the various contexts cause application objects to adapt their operations continuously. This adaptation also affects communication, thereby demanding a seamless interoperation between context detection, triggering, and communication establishment mechanisms. Third, the heterogeneity of devices, network technologies, network types, integration with the physical objects, and the breadth of functional heterogeneity make achieving interoperability among application objects difficult.[5] Fourth, a set of core reusable services (such as service discovery, resource management, and object persistence) is usually required for any system support to provide more practical solutions for distributed collaboration.

Some notable client-server middleware technologies for both enterprise and mobile networks support an asymmetric communication model in which the ORB propagates a method invocation request from a client to a particular server object. Clearly, connecting to a server implies that a client must discover the server dynamically or use prior information. Application objects executing in ad hoc environments are completely autonomous and thus, characterizing them as either clients or servers is hard. Moreover, there is no guarantee that they have any prior knowledge of each other. We need a symmetric communication model that lets a pair of remote objects communicate without really knowing or seeing each other. Other middleware technologies promote such communication models. For example, Lime[15] and Tspaces[16] use tuple spaces to let objects transparently communicate with each other.

```
//context source
RCSMContext dc {
        char [] string location;
        boolean light;
};

//beginning of context-sensitive interface
interface instructor_object {

//context variables
        RCSMContext_var dc C1
            where location="screen";
        RCSMContext_var dc C2
            where light=true;
        RCSMContext_var dc C3
            where light=false;

//context-sensitive method
        [outgoing]
        [activate when C1^(C2->C3)]
        void distribute (string lectures);
};

//end of context-sensitive interface
```

Figure 3. A context-sensitive interface that uses CA-IDL in a classroom scenario.

We prefer a message-oriented and application-transparent communication model because it lets us take a publish-subscribe approach, easily support more complex communication patterns, and allow context-sensitive communications. We adopted this approach in RCSM and call it a context-triggered communication model. The RCSM ORB supports communication in this fashion. However, having a context-triggered communication protocol, which is internal to the R-ORB, does not suffice. We still need a good "packaging" solution to uniformly ease both the development and runtime aspects between the context-processing layer (the ADCs) and the communication layer (the context-triggered communication mechanism). We chose this (the CORBA, ORB-oriented) approach because the ORB functions as a single gathering point for

- Appropriately isolating the intricacies of various transport protocols from the application objects by providing platform-independent interfaces
- Providing uniform interfaces to deploy and use various distributed services in the middleware for enhancing the application objects' capabilities
- Providing reusable mechanisms to facilitate object and service discovery, object registration, object activation and method invocation, and data marshalling over distributed environments

Without an ORB, these capabilities are difficult and often cumbersome to provide in the transport layer, operating system, and application layer.

Context-triggered point-to-point communication channel management. A context-triggered communication channel (CTC) between two devices is a type of communication link between two remote R-ORBs. Such an inter-ORB link is established and terminated primarily based on application-specific context specification. During application execution, the R-ORB performs proactive device discovery and uses its R-GIOP (RCSM General Inter-ORB Protocol) to establish and maintain a CTC with a remote device. The protocol establishes a new channel based on the following conditions:

- *Reachability*. Two devices—*A* and *B*—exist such that they are in each other's transmission range.
- *Existence of context-aware objects*. A and B each have at least one context-sensitive object.
- *Suitability of activation in the current context*. There exists at least one ADC component in both *A* and *B* that has recently generated a context-match event, implying that the application object associated with such an ADC has at least one method that can be invoked currently based on its context specification. Let these objects be O_a and O_b and their methods be M_a and M_b for *A* and *B*, respectively.
- *Compatibility*. M_a and M_b are suitable to exchange data with each other in the sense of matching interface signatures, including the number and types of parameters,

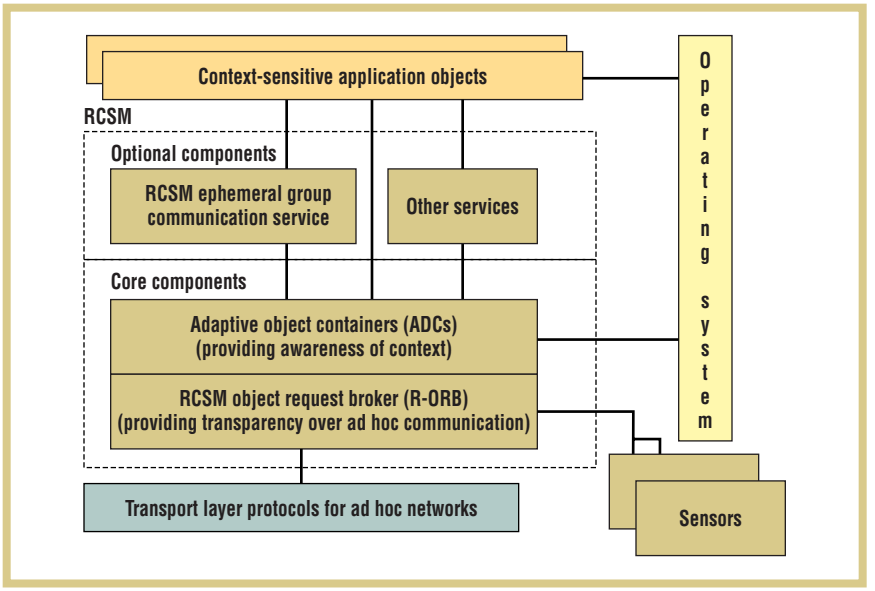

Figure 4. RCSM's integrated components.

or other application-specific criteria, such as compatibility of radio-frequency identifiers, security attributes, and so on.

To perform channel establishment, the R-ORB provides an adaptive device and service discovery mechanism. As such, if no object in the device is currently suitable to be activated, then the device and service discovery services are completely turned off to save bandwidth.

Ad hoc ephemeral group communication using the R-ORB. In the second scenario example described earlier, students form dynamic groups only for the class's duration, and the instructor joins and leaves these groups for even shorter durations. As opposed to groups in traditional distributed systems, these student groups are dynamically formed and ephemeral. We call this kind of group a *context-aware ephemeral group*. CAEG management and communication use context as the collaboration agreement among several devices. The CAEG communication mechanism is an optional module of RCSM and relies on the R-ORB for ad hoc communication and on the ADCs for context analysis.

RCSM component integration

Figure 4 shows how all of RCSM's components are layered inside a device. The R-ORB assumes the availability of reliable transport protocols; only one R-ORB per device is sufficient. The number of ADCs depends on the number of context-sensitive objects in the device. ADCs periodically collect the necessary "raw context data" through the R-ORB, which in turn collects the data from sensors and the operating system. Initially, each ADC registers with the R-ORB to express its needs for contexts and to publish the corresponding context-sensitive interface. RCSM is called reconfigurable because it allows addition or deletion of individual ADCs during runtime (to manage new or existing context-sensitive application objects) without affecting other runtime operations inside RCSM.

To show how the integrated components in RCSM work together, consider the first scenario in the earlier example. When the instructor is near the screen, the ADC generated from Figure 3's interface finds that the new location is *screen*. At the same time, the light intensity is high, thus the ADC sees that light value as *true*. Then, the instructor turns off the light in the classroom at the beginning of the lecture, so the ADC sees this light value as *false*. Thus, the context expression specified in Figure 3, C1 ^ (C2 → C3), is found to be true. The ADC conveys this context-match event immediately to the R-ORB, which then looks for a communication partner—that is, a remote context-aware object that has a matched interface. When the R-ORB finds that partner (an object on a student's PDA, for example), it sets up a communication channel with that partner and notifies the ADC. The ADC then activates the associated action to download the materials. Both objects in the instructor's and the students' PDAs are invoked transparently through their respective ADCs while the R-ORB addresses the actual information exchange. This, in effect, achieves transparent and peer-to-peer interactions.

To evaluate RCSM, we are currently implementing a project called Smart Classroom as a testbed. Smart Classroom monitors the classroom context (location of students or instructor, noise, light, and mobility) and uses this context to trigger communication activity among the students and the instructor. Related work in this area is based on different infrastructures and has different objectives. For example, Smart Kindergarten uses sensor data on children or toys to make a record for the instructor to review children's activities and track their learning progress.[25] Smart Kindergarten analyzes and stores context data in a wired infrastructure. Classroom 2000 captures classroom context (teaching material or student notes) to automatically generate Web-accessible multimedia class files for the instructor and students.[26] Both projects convert context information and the analysis of context to permanent data (a record or an output file) for later use—neither uses context dynamically.

All the RCSM components in Figure 4 could be implemented in software. However, we have chosen a software-hardware codesign approach to explore the possibility of providing some of RCSM's functionality in commodity hardware cards and to identify the unique issues related to RCSM's hardware implementation. We are also building a software-only version of RCSM to quantify the potential performance benefits of the software-hardware codesigned

Figure 5. A PDA, equipped with our current RCSM prototype, for the Smart Classroom testbed. A software-only version of RCSM is currently under development.

version. In the software-only version, the ADCs, CAEG, and R-ORB run as separate processes on a Windows CE operating system. In the software-hardware codesigned version, the R-ORB runs on hardware. Field programmable gate arrays facilitate our hardware design, because we can easily load alternative gate-level designs of the R-ORB onto the same FPGA without fabricating custom hardware. We are using the Celoxica DK-1 tool to write the R-ORB behavior in HandelC, a parallel version of the C language. The VHDL (or Verilog) code generated by DK-1 is fed into a Synopsis FPGAExpress synthesis tool to generate the FPGA's gate-level design.

In our current design, each node in the testbed has the configuration in Figure 5, which consists of the following components: a Casio E-200 PDA; a Radiometrix radio packet controller (RPC); a Trenz Electronic USB-compatible Xilinx Spartan II FPGA board; a noise, light, and motion sensor; and a location-tracking component. We have also developed simple location-beaconing boards by integrating an FPGA with the RPC controller. The current CA-IDL compiler generates the ADCs in C++ for the Windows CE operating system. Each ADC takes on the average 3 Kbytes. The R-ORB's initial hardware prototype runs on a Spartan II FPGA. A particular version of the R-ORB designed to support 16 application objects (with 16 context-sensitive methods) and 16 concurrent communication channels currently occupies approximately 31,000 logic gates in this FPGA. The FPGA with this particular version of the R-ORB has a maximum clock cycle of 150 MHz. In each cycle, this version of the R-ORB can process 1 byte of data to and from the application layer and other devices. The current design only represents one particular implementation of RCSM. Figure 6 shows the future configuration, which will have a much more compact layout and can be used as a plug-in module on a PDA.

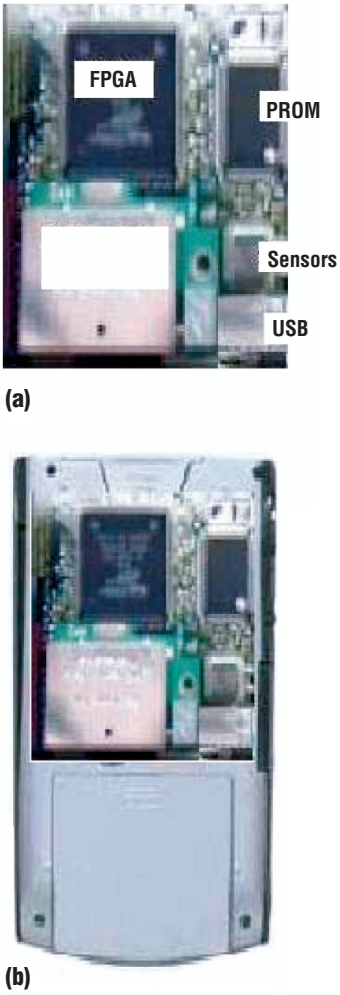

Figure 6. (a) An expected future RCSM prototype and its use as a plug-in module at the (b) back of a PDA.

ACKNOWLEDGMENTS

This research is supported in part by the National Science Foundation under grants ANI-0123980 and ANI-0196156. Microsoft Research and Tektronix donated part of the equipment used in the testbed's development. We thank the anonymous reviewers for their comments and suggestions. We especially thank Guerney Hunt of IBM T.J. Watson Research for providing valuable assistance in improving this article. We also appreciate the assistance of Siddharth Seth, Pavankumar Nallamothu, and Deepak Chandrasekar in the development of the testbed.

REFERENCES

1. M. Weiser, "The Computer for the Twenty-First Century," *Scientific Am.*, vol. 265, no. 3, Sept. 1991, pp. 66–75.
2. S.K.S. Gupta et al., "An Overview of Pervasive Computing," *IEEE Personal Comm.*, vol. 8, no. 4, Aug. 2001, pp. 8–9.
3. G. Abowd and E.D. Mynatt, "Charting Past, Present, and Future Research in Ubiquitous Computing," *ACM Trans. Computer Human Interaction*, vol. 7, no. 1, Mar. 2000, pp. 29–58.
4. G. Chen and D. Kotz, *A Survey of Context-Aware Mobile Computing Research*, tech. report TR2000-381, Dept. of Computer Science, Dartmouth College, Hanover, N.H., 2000.
5. T. Kindberg and A. Fox, "System Software for Ubiquitous Computing," *IEEE Pervasive Computing*, vol. 1, no. 1, Jan.–Mar. 2002, pp. 70–81.
6. D. Garlan et al., "Project Aura: Toward Distraction-Free Pervasive Computing," *IEEE Pervasive Computing*, vol. 1, no. 2, Apr.–June 2002, pp. 22–31.
7. M.L. Dertouzos, "The Future of Computing," *Scientific Am.*, vol. 281, no. 2, Aug. 1999, pp. 52–55.

8. B. Schilit, N. Adams, and R. Want, "Context-Aware Computing Applications," *Proc. IEEE Workshop Mobile Computing Systems and Applications*, IEEE CS Press, Los Alamitos, Calif., 1994, pp. 85–90.

9. A.K Dey, "Understanding and Using Context," *J. Personal and Ubiquitous Computing*, vol. 5, no. 1, Feb. 2001, pp. 4–7.

10. S.S. Yau and F. Karim, "Context-Sensitive Middleware for Real-Time Software in Ubiquitous Computing Environments," *Proc. 4th IEEE Int'l Symp. Object-Oriented Real-Time Distributed Computing*, IEEE CS Press, Los Alamitos, Calif., 2001, pp. 163–170.

11. N. Marmasse and C. Schmandt, "Location-aware Information Delivery with comMotion," *Proc. 2nd Int'l Symp. Handheld and Ubiquitous Computing* (HUC 2K), Lecture Notes in Computer Science, P. Thomas and H.-W. Gellersen, eds., vol. 1927, no. 1, Springer-Verlag, Berlin, 2000, pp. 157–171.

12. P.A. Bernstein, "Middleware: A Model for Distributed Services," *Comm. ACM*, vol. 39, no. 2, Feb. 1996, pp. 86–97.

13. K. Geihs, "Middleware Challenges Ahead," *Computer*, vol. 34, no. 6, June 2001, pp. 24–31.

14. S.S. Yau and F. Karim, "Context-Sensitive Object Request Broker for Ubiquitous Computing Environments," *Proc. 8th IEEE Workshop Future Trends Distributed Computing Systems*, IEEE CS Press, Los Alamitos, Calif., 2001, pp. 34–40.

15. A. Murphy, G. Picco, and G.-C. Roman, "LIME: A Middleware for Physical and Logical Mobility," *Proc. 21st Int'l Conf. Distributed Computing Systems*, IEEE CS Press, Los Alamitos, Calif., 2001, pp. 524–533.

16. T.J. Lehman et al., "Hitting the Distributed Computing Sweet Spot with TSpaces," *Computer Networks*, vol. 35, no. 4, Mar. 2001, pp. 457–472.

17. C. Mascolo et al., "XMIDDLE: A Data-Sharing Middleware for Mobile Computing," *J. Wireless Personal Comm.*, vol. 21, no. 1, Apr. 2002, pp. 77–103.

18. M. Haahr, R. Cunningham, and V. Cahill, "Supporting CORBA Applications in a Mobile Environment," *Proc. 5th Int'l Conf. Mobile Computing and Networking* (Mobicom 99), ACM Press, New York, 1999, pp. 36–47.

19. C. Hess, M. Roman, and R.H. Campbell, "Building Applications for Ubiquitous Computing Environments," to be published in *Proc. Int'l Conf. Pervasive Computing*, 2002; http://choices.cs.uiuc.edu/gaia.

20. Y. Nakanishi et al., "Context-Aware Messaging Service," *J. Personal and Ubiquitous Computing*, vol. 4, no. 4, Sept. 2000, pp. 221–224.

21. N. Sawhney and C. Schmandt, "Nomadic Radio: Speech and Audio Interaction for Contextual Messaging in Nomadic Environments," *ACM Trans. Computer Human Interaction*, vol. 7, no. 3, Sept. 2000, pp. 353–383.

22. D.C. Schmidt et al., "Software Architectures for Reducing Priority Inversion and Non-Determinism in Real-Time Object Request Brokers," *Real-Time Systems*, vol. 21, nos. 1–2, July–Sept. 2001, pp. 77–125.

23. A.K. Dey and G. Abowd, "The Context-Toolkit: Aiding the Development of Context-Aware Applications," *Proc. Conf. Human Factors in Computing Systems* (CHI), ACM Press, New York, 1999, pp. 434–441.

24. A. Schmidt et al., "Advanced Interaction in Context," *Proc. 1st Int'l Symp. Handheld and Ubiquitous Computing* (HUC 99), Lecture Notes in Computer Science, G. Goos, J. Hartmanis, and J. van Leeuwen, eds., vol. 1707, no. 1, Springer-Verlag, Berlin, 1999, pp. 89–101.

25. A. Chen et al., "A Support Infrastructure for Smart Kindergarten," *IEEE Pervasive Computing*, vol. 1, no. 2, Apr.–June 2002, pp. 49–57.

26. G.D. Abowd, "Classroom 2000: An Experiment with the Instrumentation of a Living Educational Environment," *IBM Systems J.*, vol. 38, no. 4, Oct. 1999, pp. 508–530.

For more information on this or any other computing topic, please visit our Digital Library at http://computer.org/publications/dlib.

the AUTHORS

Stephen S. Yau is a professor in the Department of Computer Science and Engineering at Arizona State University. He has served as the president of the IEEE Computer Society and the editor-in-chief of *Computer* magazine. His research is in distributed computing systems, software engineering, mobile computing, and adaptive middleware. He received a BS from National Taiwan University, and an MS and PhD from the University of Illinois, Urbana, all in electrical engineering. He is a life fellow of the IEEE and a fellow of American Association for the Advancement of Science. Contact him at yau@asu.edu; www.eas.asu.edu/~ssyau.

Fariaz Karim is a PhD candidate in the Computer Science and Engineering Department at Arizona State University. His research interests include middleware, software–hardware codesign of embedded systems, mobile and pervasive computing, wireless networks, and component-based software development. He received a BS in computer science from Arizona State University. He is a member of the ACM, IEEE, and UPE. Contact him at karim@asu.edu; www.fariakarim.com.

Yu Wang is a PhD candidate in the Department of Computer Science and Engineering at Arizona State University. Her research interests include context-sensitive and situation-aware applications, mobile and distributed computing, pervasive computing, and software engineering. She received her BS in computer science from Wuhan University in China. She is a member of Upsilon Pi Epsilon. Contact her at wangyu@asu.edu; www.eas.asu.edu/~cse355s.

Bin Wang is a PhD student in the Department of Computer Science and Engineering at Arizona State University. His research interests include wireless networks, mobile computing, pervasive and ubiquitous computing, and group communication. He received his BEng in computer engineering and an MS in computer science from the Beijing University of Post & Telecommunications, China. Contact him at bin.wang@asu.edu; http://enws849.eas.asu.edu/~bwang.

Sandeep K.S. Gupta is an associate professor in the Department of Computer Science and Engineering at Arizona State University. His research interests include wireless networks, mobile and pervasive computing, middleware, and embedded sensor networks. He received the BTech in computer science and engineering from the Institute of Technology, Banaras Hindu University, India, his Mtech in computer science and engineering from the Indian Institute of Technology, Kanpur, and his MS and PhD in computer and information science from Ohio State University. He is a member of the ACM and a senior member of the IEEE. Contact him at sandeep.gupta@asu.edu; www.eas.asu.edu/~gupta.