

A Lightweight Middleware Protocol for Ad Hoc Distributed Object Computing in Ubiquitous Computing Environments

Stephen S. Yau and Fariaz Karim
Computer Science and Engineering Department
Arizona State University
Tempe, AZ 85287-5406, USA
{yau, karim}@asu.edu

Abstract

Devices in ubiquitous computing environments are usually embedded, wearable, and handheld, have resource constraints, and are all connected to each other through wireless connections and other computers possibly through fixed network infrastructures, such as the Internet. These devices may form numerous webs of short-range and often low-power mobile ad hoc networks to exchange information. Distributed object computing (DOC) middleware technologies have been successful in promoting high quality and reusable distributed software for enterprise-oriented environments. In order to reap the same benefit in ubiquitous computing environments, it is important to note that the natural interactions among distributed objects in ubiquitous computing environments are quite different due to various factors, such as bandwidth constraints, unpredictable device mobility, network topology change, and context-sensitivity (or situation-awareness) of application objects. Hence, the interactions among distributed objects tend to be more spontaneous and short-lived rather than predictable and long-term. In this paper, a middleware protocol, RKF, to facilitate distributed object-based application software to interact in an ad hoc fashion in ubiquitous computing environments is presented. RKF addresses both spontaneous object discovery and context-sensitive object data exchange. Our experimental results, based on RKF's implementation and evaluation inside the object request broker of our RCSM middleware test bed, indicate that it is lightweight, has good performance, and can be easily used in PDA-like devices.

Keywords: Ubiquitous computing environments, lightweight middleware protocol, distributed object computing middleware, context-sensitivity, situation-awareness, mobile ad hoc networks, Reconfigurable Context-Sensitive Middleware, context-sensitive communications.

1. Introduction

A major goal of ubiquitous computing (also known as pervasive computing and invisible computing) is to make computing unobtrusive to such a degree that the enabling technologies essentially become transparent [1]. A ubiquitous computing environment can be considered as a collection of embedded, wearable, and handheld devices, all connected to each other through wireless connections, possibly through mobile ad hoc networks. Devices in ubiquitous computing environments may form mobile ad hoc networks or connect to fixed infrastructures. This phenomenon changes the way distributed application programs discover and interact with each other. Moreover, some of these application programs may be context-sensitive in the sense that they adaptively take different actions in different contexts [2]. As such, these application programs can also engage in context-sensitive communications, where a communication channel is established spontaneously based on the respective context-sensitivity of application programs, as oppose to being client-initiated [3-5].

Distributed object computing (DOC) middleware technologies have been very effective in addressing various important challenges in enterprise environments [6]. Advances in both industry standard specifications, such as CORBA, COM (or .NET) and TINA-C, and various research prototypes, such as TAO [7], have made the DOC middleware technologies practical to facilitate the development-time and runtime operations of distributed application software. In order to make DOC middleware technologies useful in ubiquitous computing environments, it is necessary to investigate the way distributed application software interacts in ubiquitous computing environments. Without such an understanding, it may become more difficult to develop effective DOC middleware technologies for ubiquitous computing applications. We will briefly discuss the issues related to the ad hoc distributed object computing below:

S1) Support for both Context-Sensitive and Client-Server Inter-Object Communications: Devices in ubiquitous computing environments may spontaneously connect with each other through mobile ad hoc networks. The topologies in mobile ad hoc networks change dynamically and devices may not know each other *a priori*. For this type of networks, context-sensitive communication paradigm may be more appropriate due to its support for spontaneous communications [3]. On the other hand, a device may connect to a previously known computer (e.g. a file or a web server) in a fixed network using client-server interaction.

S2) Context Sensing: To provide context-sensitive communications [3] to the application software, a middleware protocol itself needs to “sense” some contexts. While other components in a middleware can provide the application-specific “context-awareness” support [8], a middleware protocol should be responsible for sensing the communication-specific contexts, such as arrival of new devices and their capabilities. In addition, it should be aware of which application objects can be activated in the current context.

S3) Spontaneous Object Discovery: Devices in ubiquitous computing environments often need to discover new objects whenever new devices join the network. In some cases, a central server (e.g. CORBA’s Trading Service) may not exist to facilitate the discovery process. As such, a middleware protocol should provide flexible support for discovering the objects in remote devices without relying on any third party or central mechanism.

S4) Short-Lived Communication: Many devices have limited energy since they are battery-powered. To help the devices prolong their operations, a middleware protocol should try to use short-lived communication channels, in which a channel is closed after data transmission is completed. This avoids the need for sending back and forth control messages to keep an existing connection alive.

S5) Efficiency and Lightweight Operation: Unlike in enterprise environments, devices in ubiquitous computing environments have diverse capabilities. A common middleware platform for all these devices may not be appropriate. A middleware protocol should be lightweight in the sense that it should be easily portable to many different types of middleware implementations without sacrificing performance or consuming resources beyond the acceptable limit.

Among the existing industry standard middleware technologies, CORBA’s General Inter-ORB Protocol (GIOP) and Microsoft’s DCOM protocol are the most prevalent. Although Java provides the Remote Method Invocation (RMI) techniques, its specification does not provide a separate protocol to the extent what CORBA or COM specifies. The Simple Object Access Protocol (SOAP), which uses the Hypertext Transfer Protocol (HTTP) and Extensible Markup Language (XML) [9], facilitates the communication of distributed web services. In addition, SOAP can potentially be used to perform distributed object computing in a limited fashion. However, these protocols are primarily more applicable to fixed networks and stationary devices. For mobile networks and ubiquitous computing, there are different middleware approaches [10-17]. For example, ALICE extends CORBA’s IOP to allow object communication for mobile server objects [15], and LIME [12] and XMIDDLE [14] assume a different interaction model based on shared tuple spaces.

In this paper, we will present a lightweight middleware protocol, called RKF, to facilitate ad hoc distributed object computing in ubiquitous computing environments, which may consist of heterogeneous networking environments, including both mobile and fixed networks. RKF can also be used as a building block to develop the communication subsystem of a distributed object computing middleware, such as an Object Request Broker (ORB). Based on RKF’s implementation and evaluation on our Reconfigurable Context-Sensitive Middleware (RCSM) [3,8] test bed, our experimental results indicate that RKF can be easily used in PDA-like devices with good performance.

2. Major Features of RKF

In this section, we will present RKF to address the issues discussed before. As shown in Figure 1, RKF can be used inside a communication subsystem (e.g. ORB) of a DOC middleware. RKF assumes the availability of transport layers for unicast and broadcast functionality. In this section, we will discuss the major features of RKF: C1, C2 and C3 to address S1, C4 and C5 to address S2 and S3, and C6 to address S4 and S5.

C1) Interface and Method Pair (IM_PAIR) as Communication Endpoints: RKF sees an object interface and each method of this object as a unique communication endpoint. We call such an endpoint an IM_PAIR. As such, an object may have many communication end points. We have chosen this

because it enables an object to associate different contexts with its different methods, thereby adding more flexibility.

C2) Incoming and Outgoing IM_PAIRS: Each IM_PAIR in RKF is considered as either incoming or outgoing. If a pair is incoming, it means that this particular IM_PAIR is not invoked until the data from a compatible and outgoing IM_PAIR is received first. From client-server point of view, the outgoing IM_PAIRs are similar to clients that initiate data communication and incoming IM_PAIRs are similar to servers that accept data communications. However, in RKF the incoming IM_PAIRs are not required to send a reply back to its incoming

partner and its host device in a mobile ad hoc network. In this sense, RKF works as an invisible mediator between two communication end points. RKF allows client-server messaging for which the client (i.e. outgoing IM_PAIR) must have the destination address of the device where the server (i.e. incoming IM_PAIR) resides.

C4) Context-Sensitive Object Information Advertisement: As we mentioned earlier, each IM_PAIR is a communication endpoint in RKF. RKF uses a broadcast-based technique to advertise the objects in the host device. However, RKF does not broadcast information about all the IM_PAIRs all the time. Instead, it only broadcasts the information

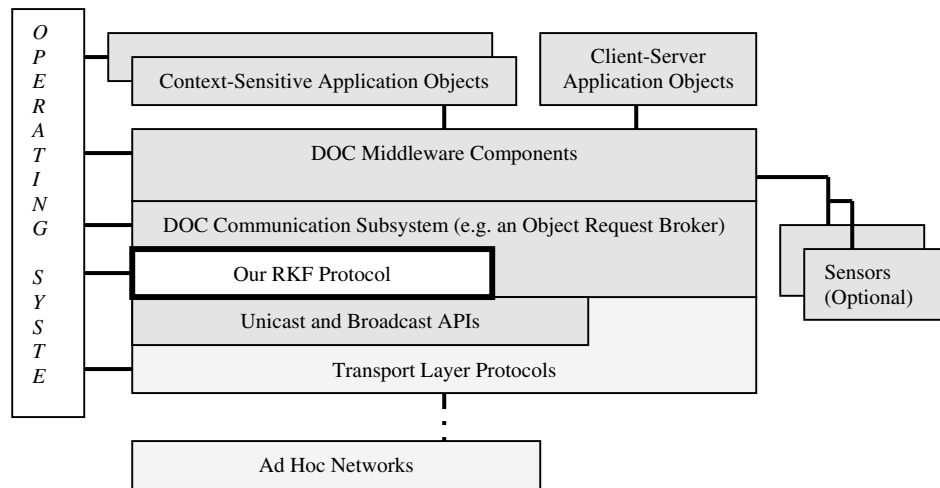


Figure 1: RKF as a building block of a distributed object computing (DOC) middleware for ubiquitous computing environments.

IM_PAIR. This allows for both unidirectional and bi-directional messages between two remote devices. Since only one end point can initiate data communication, a valid communication through RKF requires that when one communication end point is incoming, the other must be outgoing.

C3) Transparent Inter-Object Data Communication: RKF supports both client-server and context-sensitive interaction semantics [3] among distributed objects. For its context-sensitive communication capabilities, we have designed RKF to manage communication between two remote IM_PAIRs without having them really “see each other”. This allows RKF to provide communication transparency to the objects in the sense that an incoming IM_PAIR does not know where the data is coming from or an outgoing IM_PAIR does not know where it is going. This enables RKF to facilitate the discovery of the best communication

related to only those IM_PAIRs, which are of type incoming and whose contexts are valid at that instance of time. This means that RKF only advertises objects that are ready to be invoked in the current context. RKF assumes that an object notifies RKF, possibly using an event-based scheme, whenever one of this object’s IM_PAIR becomes suitable to be invoked [3].

C5) Distributed Peer-Matching: Whenever an instance of RKF receives IM_PAIR broadcast from other RKF, it performs a peer matching process to identify if any of the remote IM_PAIRs could be valid communication partners of any IM_PAIR in the host device. This peer matching is done locally in each device by comparing a list of compatible IM_PAIRs of the local IM_PAIRs. This enables RKF to discover compatible peer devices without relying on an infrastructure or centralized approach.

C6) Absence of Separate Communication Establishment and Termination Messages: After every peer matching, RKF concludes whether any remote IM_PAIR is a potential communication partner of any local IM_PAIR. If so, the RKF on the outgoing IM_PAIR side invokes the object and its method corresponding to the local IM_PAIR, retrieves the method data, and transmits a single message with these data to the device in which the new incoming IM_PAIR partner resides. The RKF on the incoming IM_PAIR side accepts the method data as an indication of both connection establishment and data. This helps RKF to reduce communication overhead.

3. Operations of RKF

In this section, we will discuss our approach to developing RKF by explaining how it works as an integral part of a DOC middleware. RKF cooperates with other components, such as ORB and object adapter, inside a DOC middleware to facilitate both client-server and context-sensitive communications. The nature of the cooperations are different as shown in Figures 2(a) and 2(b). In client-server communications, the cooperation as shown in Figure 2(a) is simple and mainly involves two phases. The cooperation for context-sensitive communications as shown in Figure 2(b) involves three phases.

In client-server communication, RKF creates a client-server communication channel for a pair of remote objects. As illustrated by the arrowhead A in Figure 2(a), a client object provides RKF with the server object's interface and the method of interest (i.e. IM_PAIR), the actual data, and destination device's address. Based on this information, RKF opens a communication channel with the destination device. The communication channel is represented as the dotted arrowhead in Figure 2(a). The RKF on the destination device then notifies the object adapter responsible for activating the server object to activate and invoke the appropriate method (the arrowhead B). The server object is not required to send back data in response. Rather, the server can send some messages later by using RKF to open a separate client-server communication channel. This enables RKF to provide an asynchronous way to address inter-object communications in mobile ad hoc networks.

In context-sensitive communication, RKF creates a context-sensitive communication channel between a pair of remote objects. Before creating such a communication channel, RKF first must identify the local objects that are "ready-to-be-activated" [3] in the current context. Based on this information, it

must discover remote compatible objects, which should be ready-to-be-activated in the current context. As we described before, RKF follows three phases before setting up a context-sensitive communication channel between a pair of objects. An important property of RKF is that it does not need to rely on any neighbor discovery protocol or any other network functionality (other than simple broadcast and unicast) to carry out these phases. These phases are illustrated in Figure 2(b) and described below:

Phase 1 – Context-Sensitive Object Information

Broadcast: RKF discovers new objects by listening for a particular type of RKF beacons. Moreover, RKF does not, by default, initiate an object discovery when new devices join the network. Instead, RKF only broadcasts the object information based on the ready-to-be-activated list of objects. In Figure 2(b), the P1.1-arrowheads indicate that the context-analyzers [3] notify RKF of the latest ready-to-be-activated set. The P1.2-arrowheads indicate the object information broadcasting process. Other devices, upon receiving these broadcasted messages, opportunistically initiate peer-matching process to determine the suitability of communication with the device from which the beacon came from. The contents of these broadcasted messages change as the ready-to-be-activated list changes. On the other hand, if the ready-to-be-activated list becomes empty, then RKF completely shuts off the broadcasting process until the list becomes non-empty.

Phase 2 - Peer Matching: In this phase, RKF discovers the compatible communication endpoints in remote devices based on the capability C5) mentioned earlier. Since each communication channel involves one incoming and one outgoing IM_PAIR and since an outgoing IM_PAIR initiates the object data exchange, the corresponding peer matching takes place on the outgoing IM_PAIR's side. The peer matching process is indicated in Figure 2(b) as an oval labeled with P2.1. After a successful peer match, RKF notifies the object adapter to activate the object and invoke the method. This part is shown in Figure 2(b) as the bidirectional arrowhead P2.2 to illustrate both up-call and down-calls to and from the object adapter.

Phase 3 – Object Data Exchange: In this phase, RKF transmits actual object level data from one object to another after the necessary object activations are performed and object data are retrieved. As mentioned earlier, RKF's object activation process hides the actual destination or

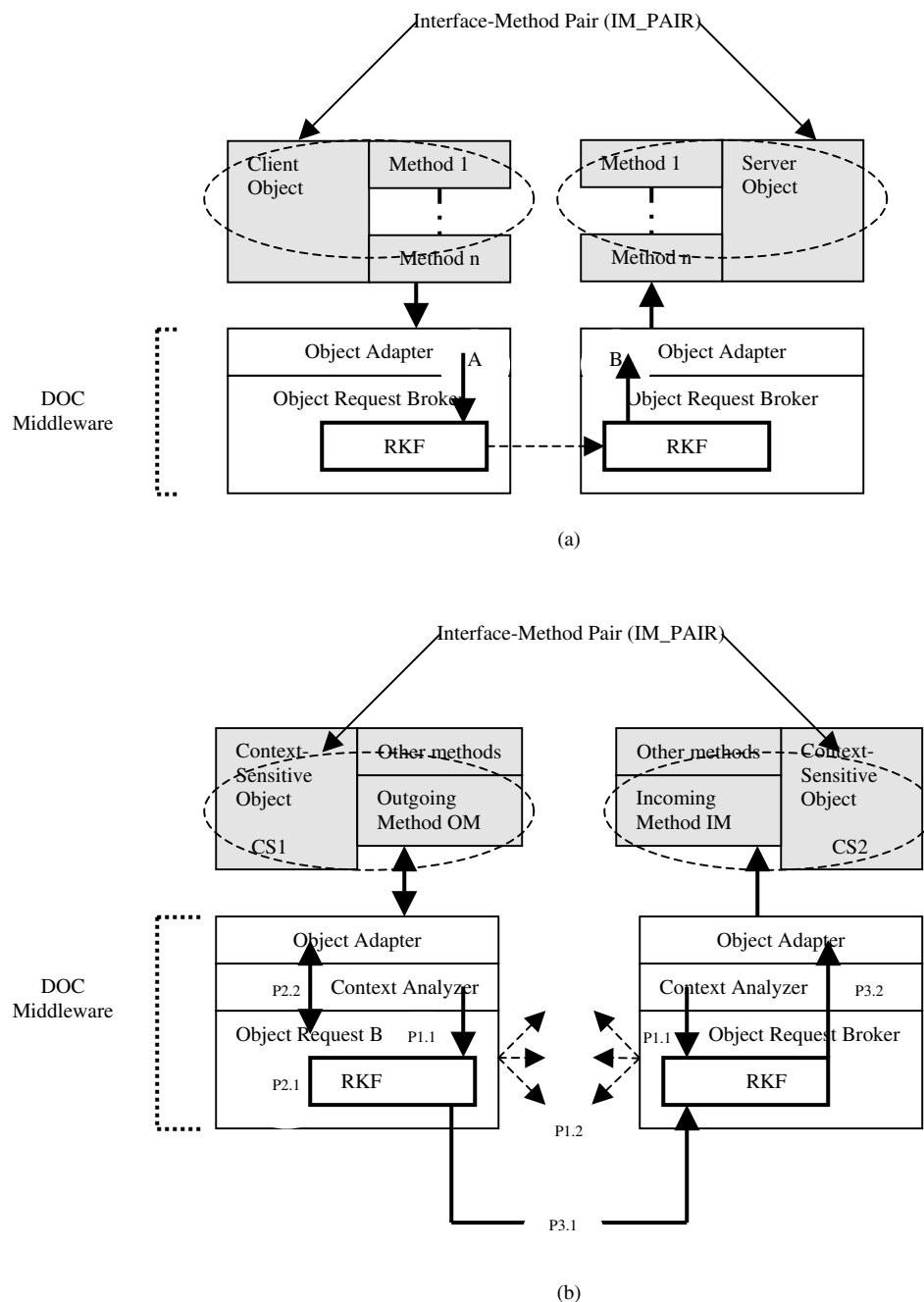


Figure 2: RKF's approach to facilitate (a) client-server and (b) context-sensitive inter-object communications in ubiquitous computing environments.

source of information following the capability described in C3). This data exchange part is shown as the arrowheads P3.1 and P3.2 in Figure 2(b).

4. RKF Implementation and Test Bed

We have implemented RKF for Windows CE 3.0 so that it can be used and evaluated inside RSCM Object

Request Broker (R-ORB) [3]. As we discussed in the last section, RKF creates a context-sensitive communication channel between a pair of objects by following three phases. However, there may be many devices in a network, and hence many pairs of potential objects may exist and require simultaneous processing of multiple channels. RKF addresses this need by running the algorithms of these phases in

parallel to create and manage multiple context-sensitive communication channels. For creating multiple communication channels, RKF has three components: R-GIOP-DOWN, R-GIOP-UP, and Context Manager, which all are executed as independent threads in infinite loops. As shown in Figure 3, the size of RKF is 5 KB for the implementation using the Microsoft Embedded C++.

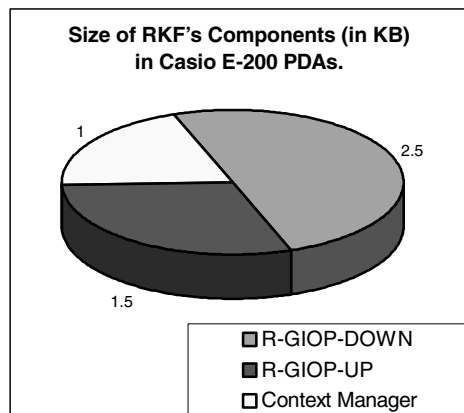


Figure 3: RKF's components. Total size of RKF is 5 KB.

RKF works as an integral part of R-ORB, which is a lightweight and context-sensitive object request broker for RCSM. RCSM provides an object-based middleware framework for context-sensitive computing in mobile ad hoc networks. R-ORB provides IM_PAIR registration, context data acquisition, raw context to structured context conversion, context data propagation, IM_PAIR method invocation, broadcast and unicast, and client-server and context-sensitive communication management facilities. R-ORB also provides the Winsock-based APIs for performing broadcast and unicast functionality. RKF uses these APIs to perform its broadcast and unicast operations in Phases 1 and 3.

We have incorporated the RKF in the R-ORB in our RCSM middleware test bed to evaluate RKF. RCSM can be configured in several different ways to provide a suitable infrastructure for ubiquitous computing experiments. At the time of our experiments, RCSM provided support for ten different contexts, including location, noise level, light intensity, and motion. As such, RCSM provided us an excellent opportunity to evaluate RKF in a real context-sensitive ubiquitous computing setting. The computing platform of our choice was several Casio E-200 PDAs, because the running versions of RCSM, including R-ORBs, were already implemented in these PDAs. Each PDA used an Intel Strong Arm

1110 with 206 MHz clock speed CPU. Each PDA was equipped with a D-LINK (Air DCF-660W Compact Flash 802.11b) adapter. These adapters were configured in mobile ad hoc network mode. As such, no infrastructure, such as an access point, was necessary to provide the communication support.

5. Experiments

The goal of our experiments is to study the performance of RKF for both client-server and context-sensitive communications. In each experiment, we programmed the threads of RKF to perform their operations in 250 MSEC intervals to coincide with the remaining R-ORB components. We used multiple PDAs equipped with the D-LINK adapters to communicate in mobile ad hoc network modes. Each PDA was programmed with multiple object-based application programs, exchanging data with each other using both client-server and context-sensitive communication channels.

Experiment 1: This experiment corresponds to the Phase 1 of RKF. Here, we are interested in RKF's performance related to its context-sensitive object information broadcast process. This latency may have an effect on how fast a DOC middleware can discover remote objects in mobile ad hoc networks. We performed the experiment by varying the number of object interfaces and their methods, and measuring the latency in 100th of nanoseconds using the *GetThreadTimes* API available in Windows CE 3.0. Figure 4 shows the result of this experiment. It is important to note that the end-to-end latency, which includes the wireless signal propagation, is beyond the control of RKF, and hence it is not shown in the results.

Experiment 2: This experiment corresponds to the Phase 2 of RKF. Here, we are interested in the efficiency of RKF during peer matching process. As we discussed in the last section, the efficiency may depend on several factors, such as the number of currently context-matched IM_PAIRs (in the CMO POOL), the number of compatible objects per interface, and the number of remote object beacons received. We performed several versions of this experiment by both varying the number of local interfaces and their methods and changing the ratio of compatible IM_PAIRs per local IM_PAIR. Figure 5 shows the result with a ratio of 16.

Experiment 3: This experiment corresponds to RKF's Phase 3. We performed two different versions of this experiment. In the first version, we were interested in

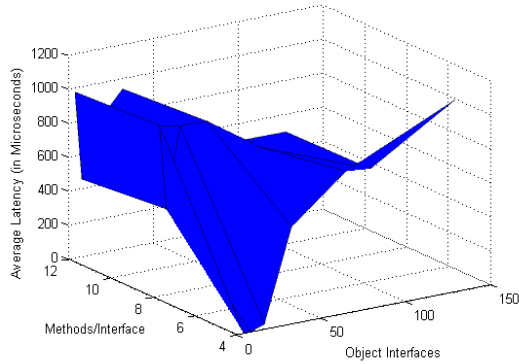


Figure 4: Context-match event processing latency in RKF during context-sensitive object information broadcast.

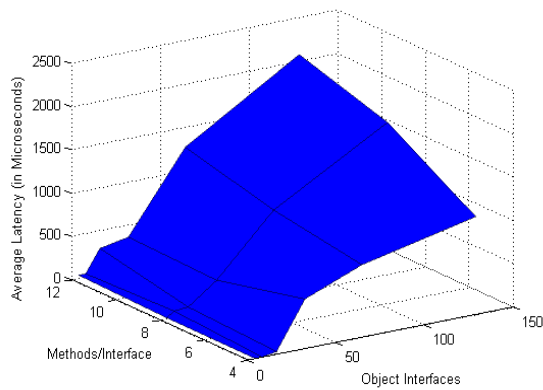
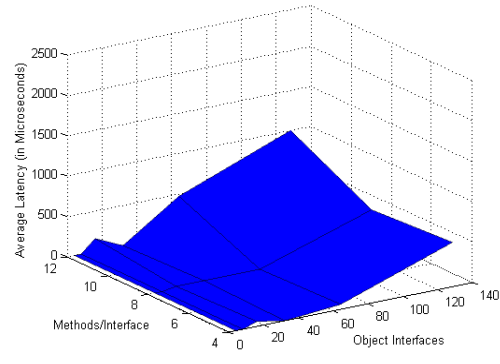


Figure 5: Peer matching latency in RKF for FRIEND_LIST/IM_PAIR = 16.

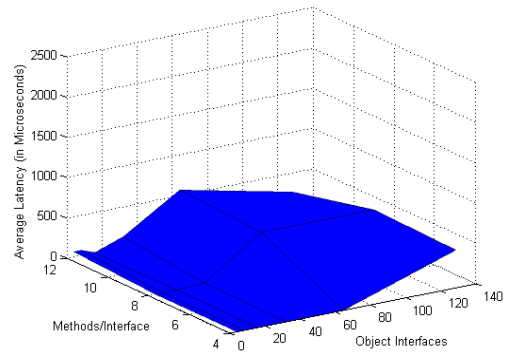
RKF's performance in processing outgoing data for context-sensitive communications. In the second version, we measured RKF's performance for processing outgoing client-server method data. Figure 6 shows these results for these two cases. Both cases show that there is no noticeable performance penalty for supporting both client-server and context-sensitive communications.

Experiment 4: This experiment corresponds to RKF's Phase 3. Unlike Experiment 3, here we are interested in measuring RKF's incoming object-data processing latency. The first version of this experiment measured the latency when the data were received using context-sensitive communication channels. Figure 7(a) shows the results of this version. In the second version, we measured the latency during the processing of incoming client-server messages. The results are shown in Figure 7(b).

Based on these experimental results, we have found that RKF performs slightly more efficiently for client-server communications than for context-



(a)



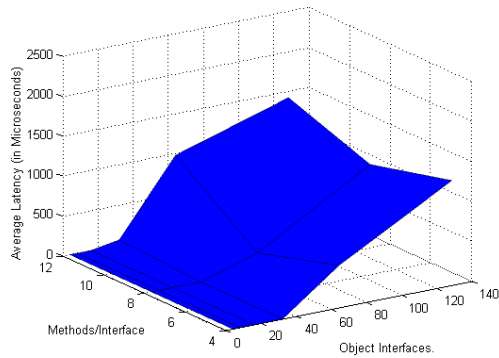
(b)

Figure 6: Outgoing IM_PAIR data processing latency during (a) context-sensitive and (b) client-server communications.

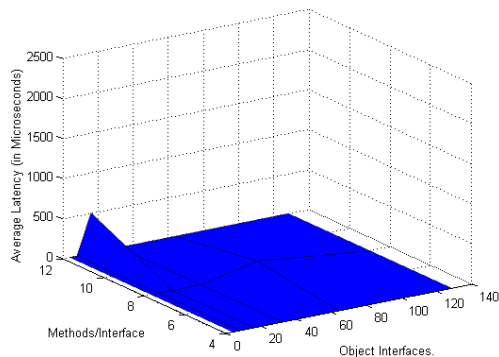
sensitive communications. The additional complexity of context-sensitive communications increases the latency. However, this increase in latency is negligible since even in the worst case the latency is in milliseconds. However, it is noted that since RKF operates in ubiquitous computing environments, the higher probability of packet collisions and loss in wireless environments may reduce the end-to-end performance of RKF significantly.

6. Discussions

We have presented RKF, which is a lightweight middleware protocol for ad hoc distributed computing in ubiquitous computing environments. RKF can be used as a building block to develop higher-level communication subsystem of a DOC middleware. RKF's support for both client-server and context-sensitive communications makes it suitable to be used in heterogeneous networking environments. Our experimental results, based on the implementation of RKF in RCSM test bed, indicate that it can be easily used in PDA-like devices. Future



(a)



(b)

Figure 7: Incoming IM_PAIR data processing latency during (a) context-sensitive and (b) client-server communications.

research along this line includes enhancing RKF to improve its energy-efficiency so that it can be even more attractive for energy-constrained devices.

Acknowledgment

This research is supported in part by National Science Foundation under grant number ANI-0123980. Microsoft Research donated part of the equipment used in the experiments. We would like to thank Deepak Chandrasekar for his assistance during our experimentation with RKF.

References

[1] M. Weiser, "The Computer for the Twenty-First Century", *Scientific American*, pp. 94-10, Sept 1991.
 [2] B. Schilit, N. Adams, and R. Want, "Context-Aware Computing Applications", *Proc. IEEE Workshop on Mobile Computing Systems and Applications*, 1994, pp. 85-90.
 [3] S. S. Yau and F. Karim, "An Adaptive Middleware for Context-Sensitive Communications for Real-Time Applications in Ubiquitous Computing

Environments", to be published in *Real-Time Systems Journal*.

[4] G. Chen and D. Kotz, "A Survey of Context-Aware Mobile Computing Research", *Technical Report TR2000-381*, Dept. of Computer Science, Dartmouth College, November 2000.

[5] N. Marmasse and C. Schmandt, "Location-aware Information Delivery with comMotion", *Lecture Notes in Computer Science*, vol. 1927, 2000, pp. 151-157.

[6] R. E. Schantz and D. C. Schmidt, "Research Advances in Middleware for Distributed Systems: State of the Art", *Communications Systems: State of the Art, Proc. IFIP World Computer Congress*, 2002.

[7] D. C. Schmidt, S. Mungee, S. F.-Gaitan, and A. Gokhale, "Software Architectures for Reducing Priority Inversion and Non-Determinism in Real-Time Object Request Brokers", *Real-Time Systems Journal*, 21(1-2), 2001, pp. 77-125.

[8] S. S. Yau, F. Karim, Y. Wang, B. Wang, and S. Gupta, "Reconfigurable Context-Sensitive Middleware for Pervasive Computing," *IEEE Pervasive Computing*, 1(3), Jul-Sept 2002, pp. 33-40.

[9] World Wide Web Consortium, Simple Object Access Protocol, version 1.1, May 2000.

[10] L. Capra, W. Emmerich, and C. Mascolo, "Middleware for Mobile Computing: Awareness vs. Transparency", Position Summary, *Proc. 8th Workshop on Hot Topics in Operating Systems*, May 2001, pp. 164-169.

[11] T. Zimmerman, "Personal Area Networks", *IBM Systems Jour.*, 35(3&4), 1996, pp. 609-617.

[12] A. Murphy, G. Picco, and G.-C. Roman, "LIME: A Middleware for Physical and Logical Mobility", *Proc. 21st Int'l Conf. Distributed Computing Systems*, April 2001, pp. 524-533.

[13] IBM Research, TSpaces Project, <http://www.almaden.ibm.com/cs/TSpaces/>.

[14] C. Mascolo, L. Capra, S. Zachariadis, and W. Emmerich, "XMIDDLE: A Data-Sharing Middleware for Mobile Computing", *Jour. Wireless Personal Communications*, 21(1), April 2002, pp. 77-103.

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

[16] IBM Research, Bluedrekar Project, <http://www.research.ibm.com/BlueDrekar/>.

[17] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt, "A Middleware Infrastructure for Active Spaces", *IEEE Pervasive Computing*, 1(4), October-December 2002, pp. 74-83.