A Context-Sensitive Middleware for Dynamic Integration of Mobile Devices with Network Infrastructures

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

Abstract

Network infrastructures (NI), such as the Internet, grid, smart spaces, and enterprise computing environments usually consists of computing nodes that are stationary, provide the backbone for environment sensing and high performance computing and communication. NI, in addition, may have various types of application software for performing resource-intensive computation. On the other hand, recent advances in the embedded systems and wireless communication technologies have increased the flexibility of using mobile devices for various practical applications. Mobile devices mostly execute application software that improves the personal productivity of the user. However, despite the rapid technology advances, mobile devices are expected to be always resource poor in comparison with the computing resources in the NIs. On the other hand, the computing resources in an NI cannot readily add the flexibility to individual users due to their fixed location and size. It is therefore desirable to combine the respective strengths of mobile devices and network infrastructures (NI) whenever possible. Dynamic integration is the process using which a mobile device can detect, communicate with, and use the required resources in nearby NIs in an application-transparent way. The benefit of dynamic integration is that the applications in both mobile device and NI can interoperate with each other as if a mobile device itself is an integral part of the NI or vice versa. In this paper, a context-sensitive middleware, called Reconfigurable Context-Sensitive Middleware (RCSM), is presented for addressing this dynamic integration problem. A novel feature of RCSM is that its dynamic integration mechanism is context-sensitive, and as such the integration between the application software in a mobile device and an NI can be restricted to specific contexts, such as a particular location or a particular time. RCSM, furthermore, provides transparency over the dynamic resource discovery and networking aspects so that application-level cohesion can be easily achieved. The integration process does not force any development-time restrictions on the application software in an NI. Our experimental results, based on the implementation of RCSM in integrated ad hoc and infrastructure-based IEEE 802.11 test bed environment, indicate that the integration process is lightweight and results in reasonably high performance in PDA-like devices and desktop PCs.

Keywords: Context-sensitive middleware, network infrastructures, mobile ad hoc networks, reconfigurable context-sensitive middleware, dynamic integration, mobile devices, and grid.

1. Introduction

The availability of increasingly inexpensive different embedded computing and wireless communication technologies, such as IEEE 802.11 and Bluetooth, is now making mobile computing more affordable [1, Chapter 7]. This affordability is also facilitating a rapid growth in the application software development for these devices. Network infrastructures (NI), such as the Internet, grid [2], and different smart-space environments usually consist of fixed computing resources that provide the backbone for environment sensing and high performance computing and communication. While mobile devices are effective in improving the productivity or efficiency of the user, NIs are needed to provide the services that are either impossible or too expensive to provide in mobile devices. However, computing nodes in NIs are usually fixed and large while mobile devices are relatively smaller and provide flexibility of accessing information and performing simple computation at anywhere and at anytime. To effectively harness the strengths of both mobile devices and NIs, it is necessary to provide a mechanism that enables a mobile device to temporarily become part of an NI in an application-transparent way. Such a mechanism is also useful in ubiquitous computing environments for supporting context-sensitive applications [3], which may sometimes adaptively need to exchange information with different computing nodes in different contexts [4-14].

Dynamic integration (DI) is the process using what a mobile device can detect, communicate with, and use the required resources in nearby NIs in an application-transparent way. A feature of DI is that it is not always necessary to statically decide the NIs to which a mobile device should be integrated with. As such, an important benefit of DI is that a mobile device can flexibly choose the best possible (application-specific) computing resource in an NI during runtime using the latest information available. As shown in Figure 1, DI may allow a mobile device to perform a task request at one computing node in an NI, retrieve the results later at a different NI, and then project the results on a display using a third computing resource in another location.

In this paper, we will present a context-sensitive middleware, called *Reconfigurable Context-Sensitive Middleware (RCSM)* [15-17], for addressing this dynamic integration problem. A novel feature of RCSM is that its dynamic integration mechanism is context-sensitive, and as such the integration between the application software in a mobile device and an NI can be restricted to specific context, such as a particular location and a particular time. RCSM, furthermore, provides transparency over the dynamic peer discovery and mobile ad hoc networking aspects so that

application-level cohesion can be easily achieved. The integration process does not force any development-time restrictions on the application software in an NI. Our experimental results, based on the implementation of RCSM in integrated ad hoc and infrastructure-based IEEE 802.11 test bed environment, indicate that the integration process is lightweight and results in reasonably high performance in PDA-like devices and desktop PCs.

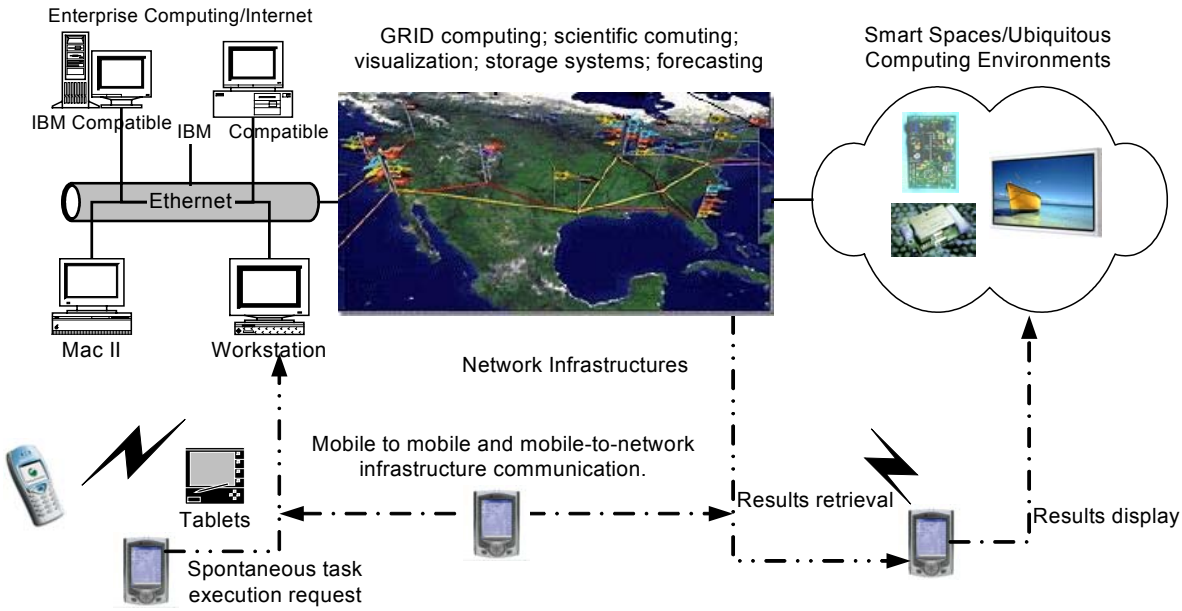


Figure 1. The scope of dynamic integration of mobile devices with network infrastructures.

2. Related Work

It is not difficult to find research prototypes or industry standards that address the mobile-infrastructure integration issues. Among the notable work, Mobile IP, Indirect TCP, and Snooping TCP [1, Chapter 10] addresses the integration issue in the network and transport layers. Mobile IP, in addition, takes advantage of local and foreign agents to track the mobility of a device in between different integrations. The Wireless Application Protocol (WAP), on the other hand, specifies a set of protocols and execution environments for mobile devices to access WWW contents [1, Chapter 11] over heterogeneous network environments. ALICE [20], XMIDDLE [9], LIME [21], TSpaces [22], Bluedrekar [23], Mobiware [24], and GAIA [7] address various aspects of mobile application integration by taking a middleware-oriented [25-27] approach. ALICE modifies the CORBA architecture [28] by introducing a mobility layer between the transport and the CORBA IIOP layers to support both mobile clients and mobile servers. The LIME middleware 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. Among the Java-spaced approaches, TSPACES focuses on a Java-based communication middleware based on the concept of tuple-spaces as well. BlueDrekar is a middleware to provide a protocol layer and a set of APIs to enable communication among Bluetooth-equipped devices. The Mobiware provides the facilities for managing an open, active, and adaptive mobile network. Mobiware accomplishes its goal by taking advantage of a CORBA-based architecture 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. Other notable projects in the related areas, such as Ninja [15] and EasyLiving [16], have broader scopes and different focus. For example, Ninja allows service composition over the Internet, but this project has not looked at how the same can be achieved in ubiquitous computing environments.

The principal difference between the existing work and our work on Reconfigurable Context-Sensitive Middleware (RCSM) [17-19] is that most existing work focus on schemes that addresses either mobile-to-mobile or mobile-to-NI communication issues, but not both. Our RCSM provides a single-point solution not only for addressing mobile-mobile and mobile-infrastructure integration but also for facilitating the development of applications that need to interoperate with each other in heterogeneous network environments. In addition, RCSM's context-sensitive property can be utilized to control the integration in specific contexts. Although in this paper we only focus on the mobile-NI integration, RCSM provides other useful services for mobile computing, especially for context-sensitive and situation-aware applications in ubiquitous computing environments [4].

3. Challenges in Dynamic Integration

In this section we will explore the challenges in performing DI by studying a simple example. Consider a scientist who monitors and analyzes the seasonal weather patterns in different parts of the world. On a particular day, he collects new data from a specific region, and initiates a modeling and forecasting computation in his laboratory PC, which is connected to a nationwide grid network. Since this particular forecasting computation takes multiple hours, he takes his PDA and continues on with his daily routine of teaching in a nearby university. While he is on the university campus, his PDA senses from the ambient environment that the analysis, which he initiated in the laboratory, is now complete. The PDA then downloads the summary of the data and notifies the scientist using a pop-up window. The PDA also connects to a nearby plasma screen and projects the analysis reports visually for the

scientist to study. The scientist then uses the stylus on the plasma screen to take down notes. These notes are then automatically downloaded to the PDA. The scientist then types further instructions for analysis on his PDA, which then delegates these instructions to his laboratory PC.

In this example, the PDA represents a mobile device. The grid network, the scientist's PC in his laboratory, and the plasma screen on the university campus represent computing resources of different NIs. This example points to the following issues:

S1) Context-sensitivity: The scientist's PDA is context-sensitive because it can sense the available computing resources and the fact that the results of the analysis are ready on the university campus. Being aware of the environment also enables the PDA to find a plasma screen nearby for visually projecting the analysis data. On the other hand, the PDA should refrain itself from finding a plasma screen off campus. Although some of these operations are application-specific, an underlying system support must be present in the PDA to enable its context-awareness.

S2) Resource availability advertisement and discovery: In order for the PDA to be aware of the fact that there is a plasma screen nearby, the plasma screen must first make itself available through an advertisement process. On the other hand, the plasma screen should be able to discover the scientist's PDA to transfer the notes. However, in this case since the PDA is mobile and may move to other locations with the user, the plasma screen may not rely on a central infrastructure to discover the PDA. This requires an ad hoc resource discovery framework that can be equally effective for fixed computing resources and mobile devices.

S3) Mobile ad hoc communication and interoperability: After the PDA detects the plasma screen, the PDA must transfer the analysis results to the display controller of this screen. On the other hand, the PDA should transfer further instructions to the grid network for further analysis of the data. To facilitate such applications that work well across mobile devices and NIs, it is necessary to provide interoperability support. This support should be present without constraining the application developer into a specific programming language or operating platform.

S4) Application-specific services: An important aspect, which we did not illustrate in the example, concerns application-specific issues, such as security and fault tolerance. Only the scientist's PDA should be able to download the analysis results from the infrastructure. In addition, the plasma screen should not allow arbitrary PDAs to start projecting data onto itself. If the PDA runs out of battery power or gets lost, the analysis results from the grid network should be recoverable through the scientist's laptop. Hence, the support to develop such services should be present.

4. Novel Features of Reconfigurable Context-Sensitive Middleware (RCSM)

RCSM is a context-sensitive middleware, which provides both development- and run-time support for application software that need context-awareness capability [17,18]. RCSM's development-time capabilities facilitate the development of application programs that work cooperatively over mobile ad hoc network environments. On the other hand, RCSM's runtime execution and communication support allows application software in multiple devices to adaptively manage their interactions with other software in changing contexts. Specifically, the following capabilities enable RCSM to address the challenges in Section 3. M1) addresses S1), M2) addresses S2) and S3), and M3) satisfies S4).

M1) OMG's Object Management Architecture (OMA)-Based Design: We have designed RCSM to follow the basic design principles of the OMA framework (OMG 1997). This is done to make RCSM both practical and easy to use with other industry standard middleware implementations. Similar to OMA and as shown in Figures 2(a) and 2(b), we represent user-level application software as application objects. However, unlike OMA, RCSM supports two types of application objects: context-sensitive and client-server. Figures 2(a) and 2(b) also illustrate that similar to OMA's Object Request Broker, RCSM's R-ORB is the component that enables application objects, implemented in different languages, to communicate in distributed environments consisting of heterogeneous networks and platforms. In addition to providing client-server communications, R-ORB facilitates context-sensitive communications among distributed objects. Figure 2(b) also indicates how the basic RCSM framework can be used as a building block to develop various middleware-specific services, such as group communication and information dissemination.

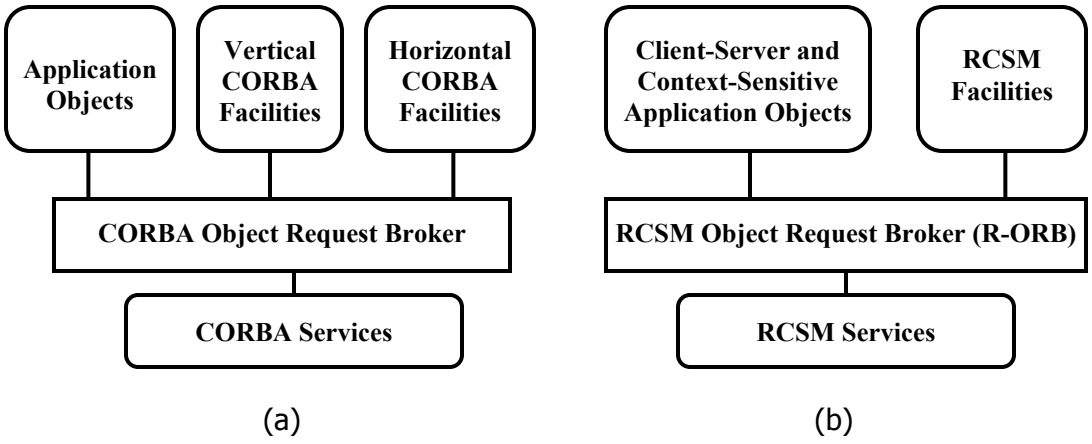


Figure 2: OMA-based (a) CORBA and (b) RCSM architectures.

M2) Support for Context-Sensitive Objects as Application Software Building Blocks: RCSM facilitates application software systems that need to be context-sensitive. In RCSM, this type of application software is represented as context-sensitive objects [17]. As shown in Figure 3, representing application software as context-sensitive objects allows developers to easily express the context-sensitivity property within the realm of object-oriented software development methodologies. The important benefit of this method is that the implementation is completely isolated from the context specification, i.e. it is context-independent. In RCSM, this is facilitated through Context-Aware Interface Definition Language (CA-IDL) [18]. On the other hand, the developer needs to focus on implementing the methods of the object in his/her favorite language without worrying about context acquisition, and in some cases, context analysis. As shown in Figure 4, object-specific context processors (OCPs) in RCSM are used to provide the object-specific context analysis support.

Similar to mature middleware standards and prototypes, such as CORBA, COM, and TAO for fixed networks, R-ORB cooperates with these OCPs to provide a run-time framework for object registration and activation, data marshalling, and request demultiplexing.

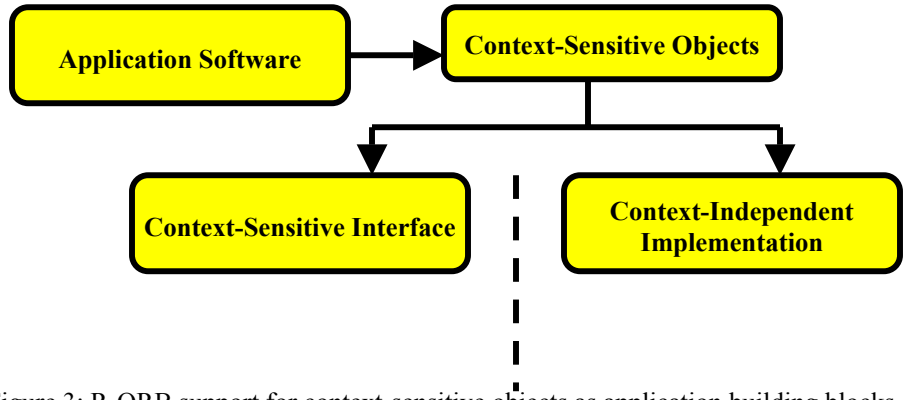


Figure 3: R-ORB support for context-sensitive objects as application building blocks.

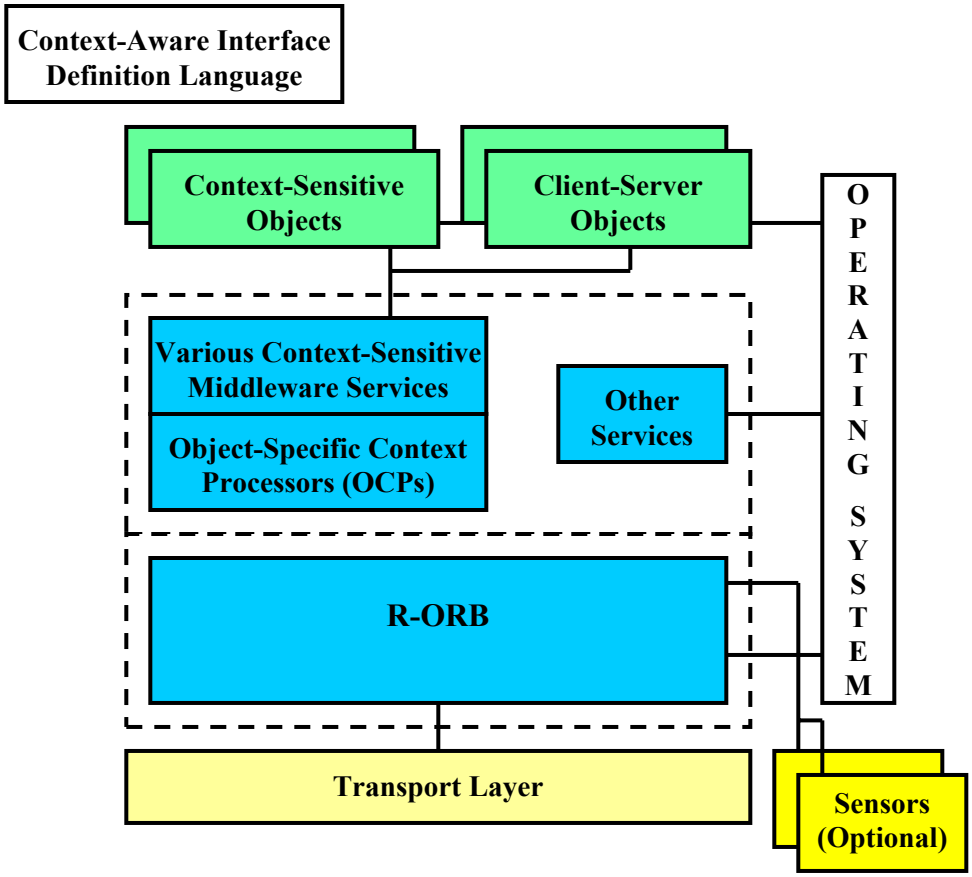


Figure 4: RCSM's architecture

RCSM also provides facilities for registering an object's interest in different contexts based on the information obtained from the interface of a context-sensitive object, R-ORB acquires various types of context attributes from different sources, such as underlying operating system, sensors, and location beacon servers. These facilities of RCSM enable the construction of higher-level and middleware-specific services that need to be context-sensitive. Furthermore, R-ORB is designed to facilitate context-triggered object activation, which lets objects take the right actions in the right context. Another feature of RCSM is to provide object-specific situation-analyzers with situation-awareness capability [19]. Situation is a set of past contexts and/or actions of individual devices relevant to future device actions. Situation-awareness is the capability of monitoring contexts, detecting situation changes and responding to the changes. This capability allows RCSM to promote wider use of context for ubiquitous computing applications.

M3) Transparency Over Context-Sensitive Communications in Mobile Ad Hoc Networks: RCSM provides transparency over context-sensitive communications for distributed context-sensitive objects that spontaneously need to exchange information with each other in changing contexts. This capability further enables these objects to rely on R-ORB for dynamically detecting the appropriate communication peers in an energy-efficient way, checking the suitability of communication establishment, and providing transparency to hide the differences in networking environments, operating systems, and hardware platforms. As shown in Figure 4, R-ORB operates above the transport layer (e.g. TCP) of a device, and hence it assumes that the necessary routing and reliable connection-oriented unicast facilities are available.

5. Key Ideas of Our Dynamic Integration Process Using RCSM

The problem of dynamic integration of a mobile device with a computing node in an NI can be characterized as follows: Establish a context-sensitive communication link between a pair of objects such that a) these objects are compatible to exchange information with each other; b) one object resides in a mobile device while the other resides in an NI; and c) both objects can be activated in the current context according to their respective context-awareness specifications. The condition in a) ensures that a compatibility check between a pair of objects is performed before establishing a communication channel. Condition b) restricts the communication endpoints to a mobile device and a computing node in an NI. Condition c) entails that before performing an integration, RCSM must first check if the

potential peer object have any further contextual restrictions (e.g. location, time, number of other devices) and then ensure that these restrictions are satisfied. For application software that is not context-sensitive, Condition c) is true by default all the time. This is especially true for NI environments because application software in these environments is usually not context-sensitive. In order to facilitate the DI process, the target environment should consist of one or more NIs and mobile devices as shown in Figure 5.

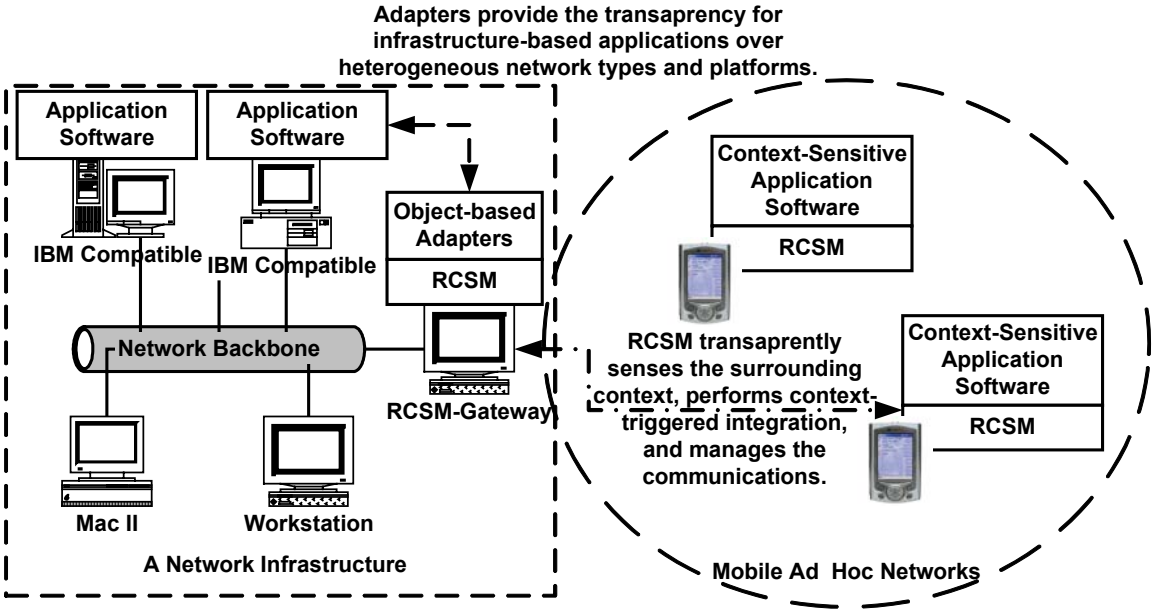


Figure 5: RCSM's context-sensitive dynamic integration support

Mobile device configuration: Each mobile device participating in the DI process should be equipped with RCSM and should have wireless ad hoc networking capabilities, such as IEEE 802.11 or Bluetooth adapters. A mobile device may have one or more application programs, implemented as context-sensitive objects. As we mentioned before, context-sensitive objects can specify the context in which they should be executed.

NI configuration: Each NI participating in the DI should have at least one computing node with both wired and wireless networking capabilities. This node, known as the RCSM-gateway, serves as an intermediary between a mobile device and a computing node in an NI. The RCSM-gateway includes an executable version of RCSM with one or more adapters. Adapters are structural design patterns for masking or enhancing the original interface of a program [29, Chapter 4]. The adapters in an RCSM-gateway work as the entry points of the application programs in

an NI. They are used to provide the transparency of communication between an application program in an NI and an object in a mobile device.

The integration process is mainly divided into the following: First, a mobile device equipped with RCSM senses the surrounding context, which corresponds to whether any of the context-sensitive objects in the mobile device can be activated in the current context, or if so, if any potential communication peer exists in the vicinity. This process takes place inside the dotted oval in Figure 3. Second, once a suitable adapter in an RCSM-gateway is discovered, the actual communication is initiated through this adapter. This process is shown in Figure 6.

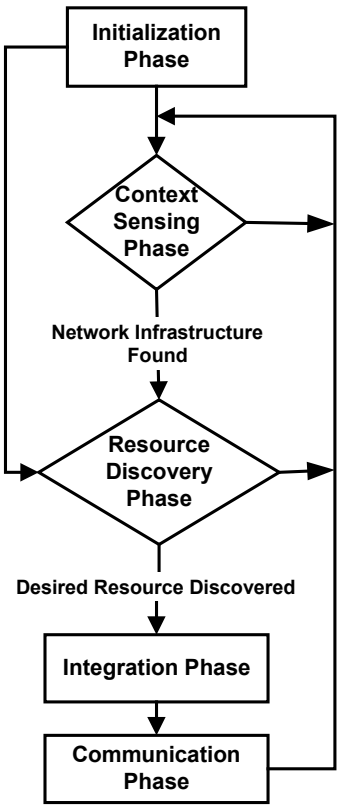


Figure 6: Dynamic integration

The principal benefit of our RCSM approach is that application programs that execute either on the mobile devices or on NIs are completely kept hidden from the system-specific DI issues. An additional benefit is that the concept of context can be customized for each application program in a mobile device. This can be used to express the preferences of application software concerning where or when to perform integration or which system to integrate with in an NI. For example, this allows a web browser to transparently integrate with a web server in a specific location at a specified time, if in fact the mobile device happens to be in that location during the specified time.

To facilitate the above approach, an application developer or a system integrator needs to take some preparatory steps for an application software in an NI that needs the DI facilities. RCSM's RKF protocol, on the other hand, addresses the actual integration during run-time. We will elaborate these in detail in Sections 6, 7, and 8. We will also discuss the implementation of our test bed and the experimental results in Sections 9 and 10.

6. Adapter Generation for an Application Program in an NI

A developer or system administrator follows the usual procedure to construct the desired application software for an NI. In order to make the application software accessible to mobile devices, the developer should perform the following activities as illustrated in Figure 7:

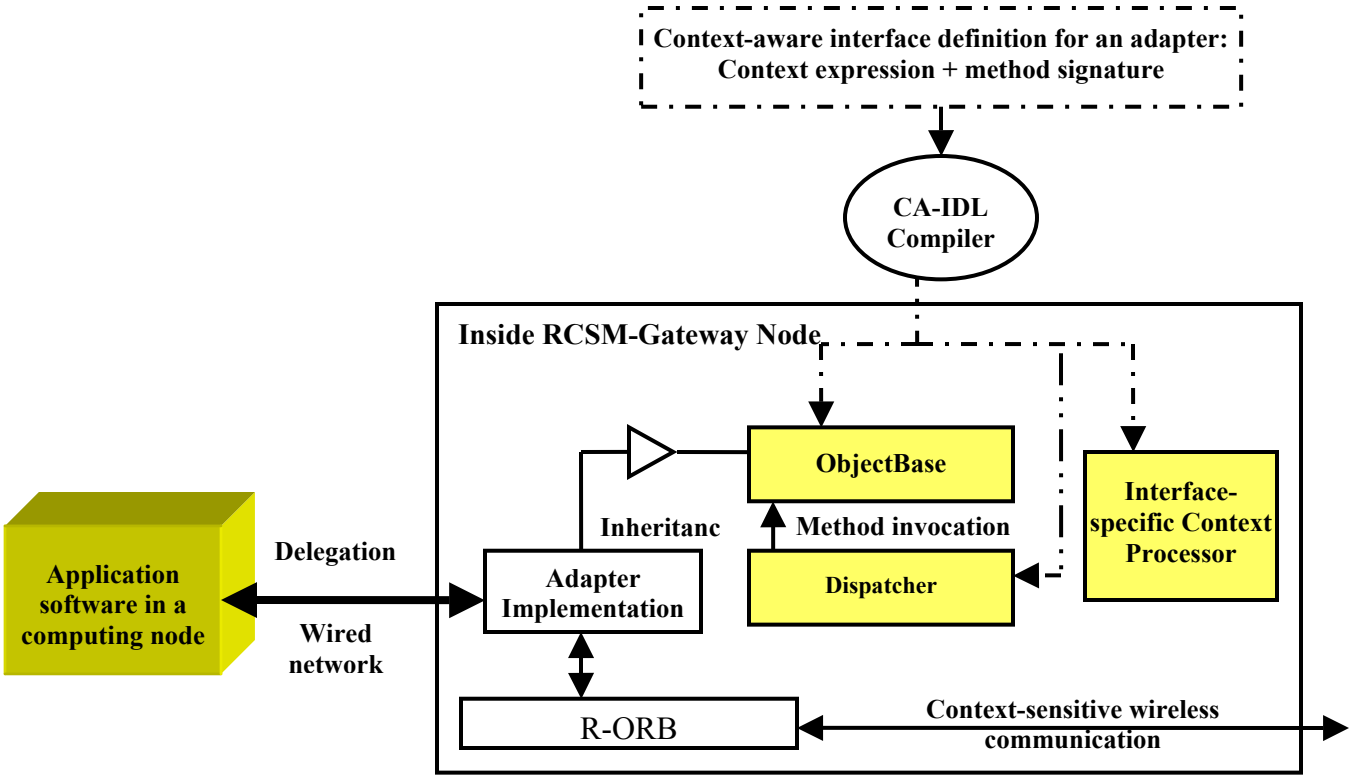


Figure 7: Adapter skeleton generation using RCSM's CA-IDL specification method for implementing the adapter to forward messages to application software.

1) Specify a context-sensitive interface for the adapter that will represent the application software. The interface consists of the method signatures that serve as the entry points to specific functions of the application program. The interface also includes a specification of a context to restrict the DI process into some specific context.

2) Use the RCSM's CA-IDL compiler to generate an object skeleton based on the interface generated in Step 1).

3) Implement the necessary delegation code inside the object skeleton. The delegation code is responsible for forwarding any task request received from a mobile device to the computing node, where the actual application software will reside. The skeleton, when completed, serves as an object adapter.

4) Register the object adapter generated in Step 3) with the R-ORB.

Note that Step 1) can be simplified if the developer does not need to restrict the dynamic integration in a specific context. Detailed information on using CA-IDL's specification method and the object skeleton generation was given in [18].

7. RCSM's Context-Sensitive Dynamic Integration Protocol:

7.1 RKF Overview:

Before presenting our integration process, we need to provide an overview of RCSM's RKF protocol. RKF is the principal component used in the integration process. It executes inside the R-ORB, and as such it functions as a main communication mechanism between a mobile device and an RCSM-gateway in an NI. In fact, we use the same protocol to facilitate context-sensitive communication between a pair of mobile devices as well. This uniformity enables the context-sensitive objects in mobile devices to seamlessly communicate with an application program irrespective of its execution platform, i.e. a computing node in an NI or a mobile device. Below, we describe the features of RKF that facilitate the DI process:

C1) Interface and Method Pair (IM_PAIR) as Communication Endpoints: RKF sees an application program as one or more context-sensitive objects. Furthermore, it 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. Since an adapter inside an RCSM-gateway has a well-defined interface, it fits well to view an adapter as a collection of IM_PAIRs.

C2) Incoming and Outgoing IM_PAIRs: Each IM_PAIR 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 the 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 IM_PAIR. This allows for both unidirectional and bi-directional messages between a pair of mobile devices and a computing node in an NI. Since only one end point can initiate data communication, a valid communication through RCSM 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 [18] 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 RCSM to provide a 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 its data is going. This enables RCSM to facilitate the discovery of the best communication peer, which may reside either in a mobile device or in a computing node in an NI. In this sense, RKF works as an invisible mediator between two communication end points. RKF allows client-server messaging where the client (i.e. outgoing IM_PAIR) must provide 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 a mobile device. In case of RCSM-gateway, RKF advertises the adapters' IM_PAIR information. However, RCSM does not broadcast information about all the IM_PAIRs all the time. Instead, it only broadcasts the information related to only those IM_PAIRs that are of type incoming and those that are ready to be invoked in the current context according to their respective context specification. The mechanism that detects this "ready-to-be-activated" set is explained in [18].

C5) Distributed Peer-Matching: Whenever an instance of RKF receives IM_PAIR broadcast from other sources, 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 RCSM 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, retrieves the result, and transmits a single message with the results to the device in which the new incoming IM_PAIR partner resides. The RKF on the incoming IM_PAIR side accepts the result as an indication of both connection establishment and data. This allows RCSM to reduce communication overhead, which is especially helpful to conserve energy in battery-powered mobile devices.

7.2 RKF's Integration Process

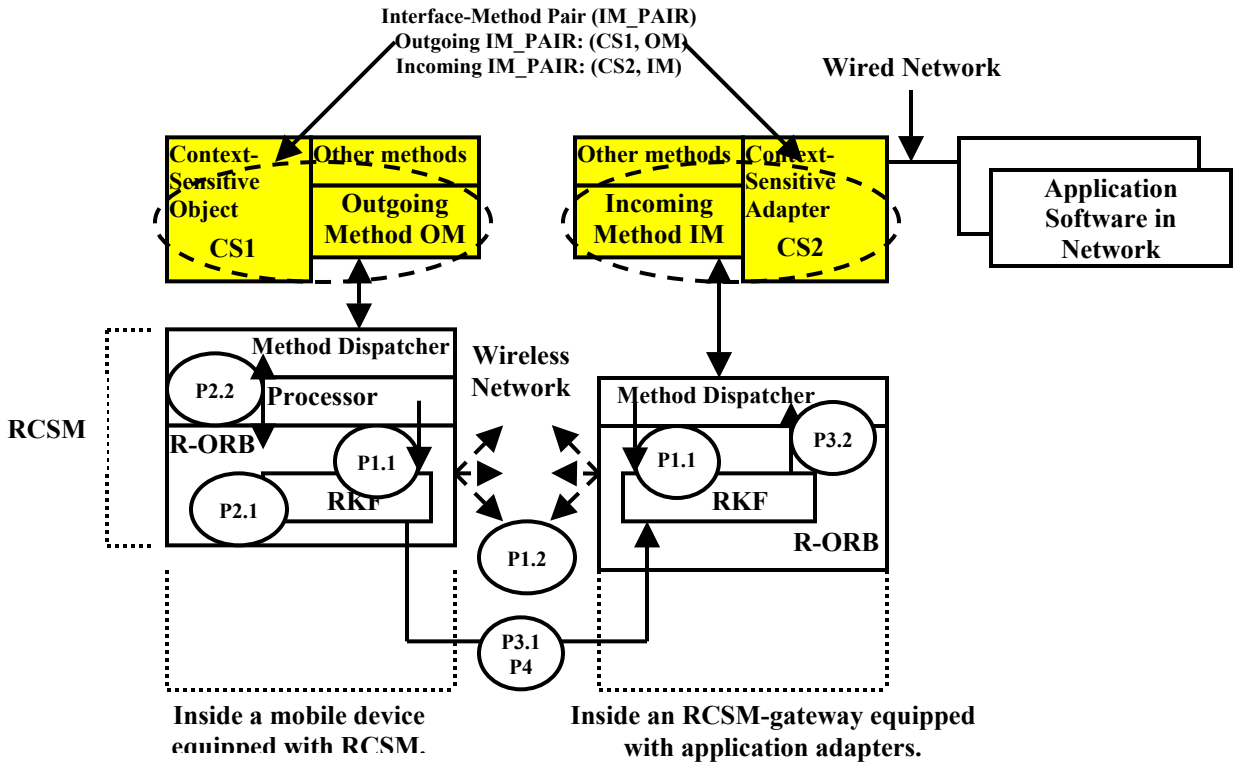


Figure 8: End-to-end view of dynamic integration using RCSM

The process of dynamic integration using RCSM is shown in Figure 8. The same process is illustrated as a system diagram in Figure 6. As shown in Figure 8, a mobile device is equipped with RCSM to be able to dynamically integrate itself with a network infrastructure. The network infrastructure, on the other hand, is connected to an RCSM-gateway server, which provides a bridge between the application software inside a network infrastructure with those in a mobile device. Each application software program in a mobile device is represented as a collection of context-sensitive objects. In case of RCSM-gateways, we use simple object adapters that work as gateways to the application programs residing on the network infrastructures. The entire process is divided into four phases, which are summarized below. All these processes are facilitated by RKF, which resides inside R-ORB.

Phase 1 - Context-Sensing: 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[1] list of objects. In Figure 8, the P1.1 arrowheads indicate that the context-processors [18] notify RKF of the latest ready-to-be-activated set. The P1.2 arrowheads indicate the object information broadcasting process. Other devices, upon hearing these broadcast messages, opportunistically initiate the peer-matching process to decide on the suitability of communication with the device from which the beacon came from. The contents of these broadcast 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 – Resource Discovery: In this phase, RCSM 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 8 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 8 as the bidirectional P2.2 arrow to illustrate both up calls and down calls to and from the object adapter.

Phase 3 – Integration Phase: Following a successful peer matching, RCSM on a mobile device notifies the RCSM inside the RCSM-gateway to activate the appropriate object adapter. The object adapter then sets up a communication link with an application program in a network infrastructure. This process is shown as the P3.1 and P3.2 arrowheads in Figure 8.

Phase 4 – Object Data Exchange: In this phase, RCSM starts transmitting actual object level data from one object to another after the necessary object activations are performed and object data are retrieved. As mentioned earlier, RCSM's object activation process hides the actual destination or source of information following the capability described in C3). This data exchange part is shown as the P4 arrowhead in Figure 8.

8. Detailed Description of the Integration Phases

To facilitate the integration process explained above, we specifically utilize the RKF protocol inside RCSM. RKF creates a context-sensitive communication channel between a pair of objects by following four phases. However, there may be many devices in nearby network infrastructures, and hence many pairs of potential objects may exist

[1] Ready-to-be-activated is a list that contains information about the local application software that can be activated in the current context. How to provide this capability in RCSM is explained in [18].

which require simultaneous processing of multiple inter-object communications. RKF addresses this need by running the algorithms that execute these phases in parallel to be able to create and manage multiple context-sensitive communication channels. In this section, we will explain the phases from the perspective of creating a single context-sensitive communication channel. For creating multiple communication channels, RKF has three components: R-GIOP-DOWN, R-GIOP-UP, and Context Manager. All components execute as independent threads in infinite loops. R-GIOP-DOWN and R-GIOP-UP are responsible for sending and receiving portions of Phases 1 and 4 respectively, and the Context Manager is responsible for Phases 2 and 3.

- **Phase 1: Context-Sensitive Object Information Broadcast**

Before discussing how RKF performs the object information broadcast, we need to present the common terms and data structures used in RKF. These terms and data structures are given in Table 1.

Table 1: Definitions of common terms and data structures in RKF for Section 6.

IM_PAIR	An {object interface, method} pair that uniquely identifies a connection point of a context-sensitive application object.
Outgoing IM_PAIR	An IM_PAIR that always initiates connection to a remote IM_PAIR for requesting some information or for initiating collaboration. A client object and one of its methods in client-server architectures is an example of an outgoing IM_PAIR.
Incoming IM_PAIR	An IM_PAIR that accepts connection from a remote IM_PAIR for exchanging some information or starting collaboration. A server object and one of its methods in client-server architectures is an example of incoming IM_PAIR.
Context-Match Event	A control message corresponding to an IM_PAIR that indicates to RKF that this pair is "ready-to-be-activated" whenever a suitable IM_PAIR partner is found in a remote device. It is not necessary for an object or any of its methods to generate these events. Object-specific context processors [18] can analyze the current context on behalf of the objects and generate these events. It is usual to have repeated context-match

	events if suitable contexts for the corresponding IM_PAIR continue to exist. The event data structure contains the corresponding IM_PAIR.
Context-Matched IM_PAIR	An IM_PAIR for which a context-match event is generated.
IM_PAIR_BEACON	A broadcast message that includes one or more IM_PAIRs and their relevant information such as interface and method signatures, XML schema, textual description, and security attributes, that may be useful for remote application objects (or middleware components to decide if this pair is a suitable communication partner. In the current implementation, one IM_PAIR message is 1 KB large.
DATA_UNICAST	A unicast message that includes the data from one object to another. RKF uses this message to facilitate transparent information exchange following a successful peer discovery.
CMO POOL	A list that contains the current local context-matched IM_PAIRs.
RO POOL	A list that contains information, received from IM_PAIR_BEACONs, of remote IM_PAIRs.
SMO POOL	A list that contains pairs of one local and one remote IM_PAIR. A pair indicates that the local IM_PAIR is compatible with the remote IM_PAIR to exchange some information.
FRIEND_LIST	This list contains a list of compatible IM_PAIRs for each local IM_PAIR. This list is used to identify the valid communication partners in remote devices.

As illustrated in Figure 9, RKF performs the following steps to spontaneously broadcast object information:

1) RKF periodically checks its CM_EVENT queue. If the queue is empty, it goes to sleep. Otherwise, it continues to Step 2).
2) If a new CM_EVENT is pending, RKF checks the validity of the corresponding IM_PAIR by checking whether the object has already registered before (RKF relies on other middleware components to accomplish this). If the corresponding IM_PAIR is valid, RKF updates the CMO POOL.

3) If the IM_PAIR is of the incoming type, it constructs an IM_PAIR_BEACON message as shown in Figure 10. RKF then uses the broadcast APIs available from the underlying transport layer to broadcast this message.

The RKF on an RCSM-gateway, upon hearing an IM_PAIR_BEACON generated in Step 3), updates its RO POOL. The same holds true when an RCSM-gateway broadcasts such beacon messages, which are then detected by an RKF in a nearby mobile device.

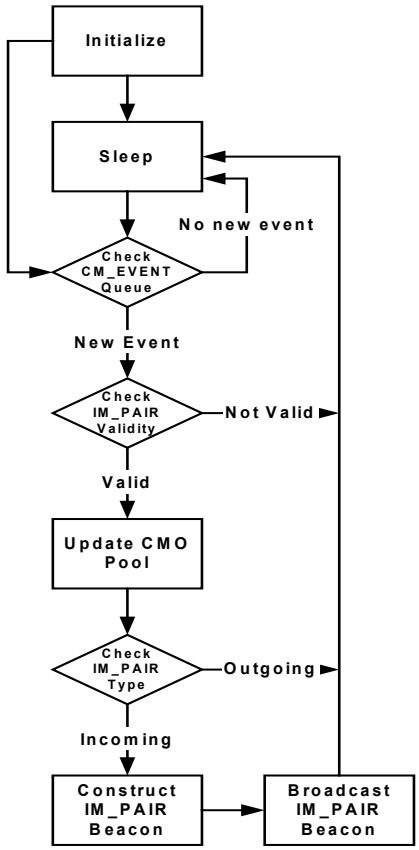


Figure 9: RKF's context-sensitive object information broadcasting procedure.

TYPE	Sender IP Address	Size	IM_PAIR Information

Figure 10: RKF's IM_PAIR_BEACON message.

- **Phases 2, 3, and 4: Peer Matching, Integration, and Object Data Exchange**

In the peer matching phase, RKF identifies which, if any, of the remote IM_PAIRs are compatible with the local IM_PAIRs that are currently ready-to-be-activated. As we mentioned earlier, it only makes sense to perform peer matching for only those IM_PAIRs, which are of the outgoing type. Performing a peer matching for incoming IM_PAIRs does not really have any effect since incoming IM_PAIRs cannot initiate data exchange. As shown in Figure 9, the peer matching includes the following steps:

1) RKF periodically checks for outgoing IM_PAIRs in the CMO POOL. If it is empty, RKF goes to sleep. Otherwise, it continues to Step 2).
2) RKF isolates the incoming IM_PAIRs from the RO POOL. In fact, the RO POOL should not contain any outgoing IM_PAIR.
3) Using the FRIEND_LIST, for each outgoing IM_PAIR from Step 2), RKF performs a matching to see if any of the remote IM_PAIRs from Step 2) appears in this list. If a match occurs, it means that a device with a compatible communication partner exists in the vicinity. In this case, RKF updates the SMO POOL with the local and remote IM_PAIRs.

Following a successful peer matching in Step 3), RKF starts the integration phase by notifying other middleware components to activate the peer-matched local IM_PAIR as represented by the last rectangle of Figure 11. The object data exchange phase is shown in Figure 12, where RKF checks the results of the invocation in the queue, and if so, constructs a unicast message destined for the device with the remote IM_PAIR. The destination IP address is retrieved from the IM_PAIR_BEACON, which RKF had previously received. Figure 12 also shows that RKF does not invoke a separate handshake protocol in order to establish a communication channel with a remote device. A remote device sees the arrival of the unicast message as an indication of a valid connection. Upon receiving a unicast, the RKF in the destination device checks whether the destination IM_PAIR is a valid local IM_PAIR. RKF checks the validity by examining whether the destination IM_PAIR exists in the CMO POOL.

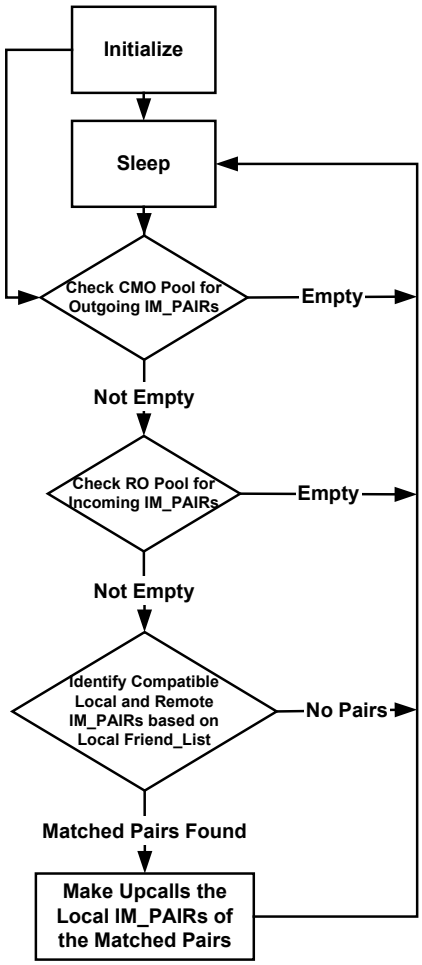


Figure 11: RKF's peer matching and integration phases.

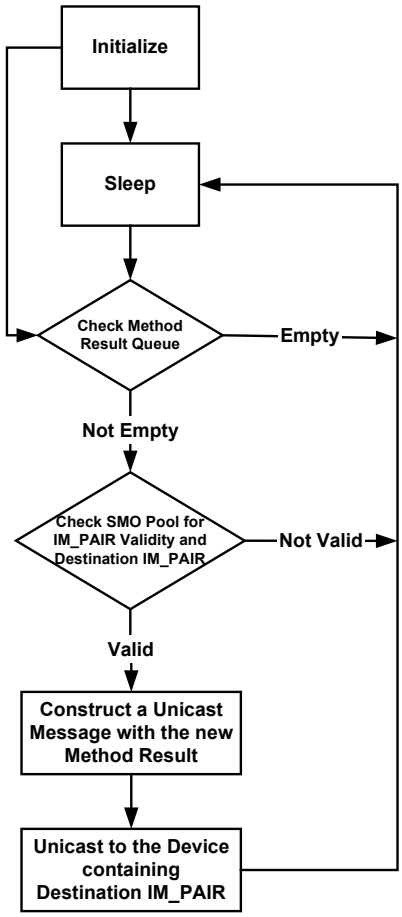


Figure 12: RKF's object data exchange phase.

9. Implementation and Experimental Test Bed

DI protocol for mobile devices: For experimentation with the mobile devices, we have implemented RKF for Windows CE 3.0 so that it could be used and evaluated inside RCSM Object Request Broker (R-ORB) [18]. RKF uses three components, all of which are implemented as threads for parallel executions. Two of these, called R-GIOP-DOWN and R-GIOP-UP, are used for processing incoming context-match events from the application layer and the IM_PAIR_BEACON messages from other devices respectively. The third thread, Context Manager, performs the peer matching and data exchange process. The implementation of RKF's total size is 5 KB and 12 KB in Casio E-200 PDAs (mobile device) and Dell Pentium PCs (RCSM-gateways).

DI protocol for NIs: For experimentation with the computing resources in an NI, we have ported R-ORB and RKF to Windows 2000 and XP-based PCs. The architecture of both RKF and R-ORB remain the same. We generated several adapters for the experimentation in Microsoft Visual Studio development environments.

Cooperation between RKF and R-ORB: RKF works as an integral part of R-ORB, R-ORB, which is a lightweight and context-sensitive object request broker for RCSM, provides an object-based framework for context-sensitive computing in 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.

RCSM Implementation on Windows CE: RCSM's implementation is divided into two parts. The first part consists of ADCs that perform object-specific context processing and analysis. The CA-IDL compiler generates a new ADC whenever a developer specifies and compiles the interface of a context-sensitive object.

The second part consists of R-ORB. R-ORB is based on a three-layered architecture as illustrated in Figure 13. Each layer consists of several objects and threads that manipulate these objects. The middle layer constitutes the code that includes all the main functionality of the R-ORB. The top and bottom layers are responsible for interfacing with the object-specific context processors and transport layer respectively. The motivation behind R-ORB's three-layered architecture is to address evolutionary changes. As shown in Figure 13, the top layer is responsible for interfacing with the other RCSM components, such as the ADCs and application programs. Responsibilities of this layer spans from accepting various messages from the upper layer and propagating them to Layer 2 through message queues. Conversely, it receives messages, which are directed toward the ADCs and application programs, from Layer 2. The Layer 3 threads perform similar operations for Layer 2 by interfacing with the Winsock socket libraries. The Layer 3 threads thus enable an R-ORB send broadcast and unicast messages to other R-ORBs. The middle layer, which called the ORB Core, consists of threads that perform all the critical functionality of the R-ORB. The layered architecture therefore allows ORB Core threads to remain independent of how R-ORB interfaces with other parts of a system. For example, it would not be difficult to have share memory-based support in R-ORB by only changing the threads in Layer 1. Another important benefit of this design each layer communicates with the other through

various upward and downward queues. The RKF protocol, which also adapts the sensitivity level of the R-ORB, executes in Layer 2. Any such adaptation systematically increases or decreases the message-processing throughput of the Layer 2. This effectively changes how many messages Layers 1 and 3 threads process. This facilitates a transparent change in the sensitivity level in these layers.

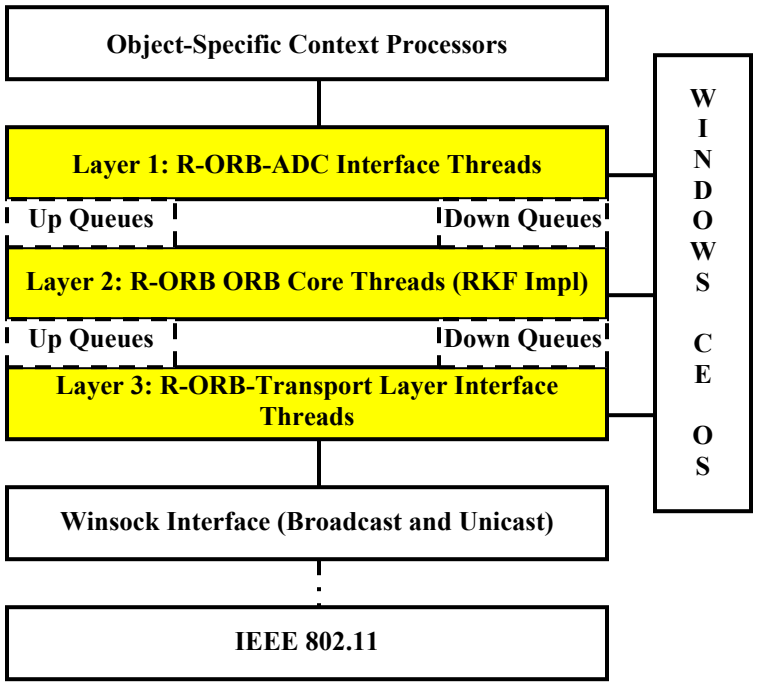


Figure 13: R-ORB's layered architecture in RCSM, in which Layer 2 includes RKF implementation.

RCSM Test Bed: We used 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 inside 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 also equipped with a D-LINK Air DCF-660W Compact Flash 802.11b adapter. These adapters were configured in ad hoc network configurations. As such, no infrastructure, such as an access point, was necessary to provide the communication support.

For the RCSM-gateway, we chose Dell Pentium PCs with both wired and wireless network adapters. For test application programs, we implemented several simple web services.

10. Experiments

The goal of our experiments is to study the performance of RKF during each phase of dynamic integration. Our primary goal was to observe the cost of DI process in mobile devices, especially due to their resource limitations. As such, in this section, we do not elaborate much on the performance of RKF in RCSM-gateways because they are executed on the desktop PCs. 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 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 modes. We conducted various runs of the same experiments by varying the following parameters: number of interfaces in a device and average number of methods per interface. We have chosen the number of interfaces as a performance indicator, because the dynamic integration mainly occurs among the interfaces of context-sensitive objects while the object implementations themselves remain hidden. We chose the second parameter because the number of methods dictates the performance during data transfer and up call mechanisms (due to method-interface lookup operation) after a device is integrated. For simplicity, we have designed each method in an interface to return 1 KB of data through a character buffer (i.e. a string). We kept this value constant for measuring the performance for both incoming and outgoing data. Each run of an experiment was executed for almost 30 minutes to get an average value.

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 middleware can discover remote objects in ad hoc networks. We performed the tests by varying the number of object interfaces and their methods, and measuring the latency in 100^{th} of nanoseconds using the GetThreadTimes API available in Windows CE 3.0. Figure 14 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 previous 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 (FRIENDS) 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 15 shows these results, from which we can see that a high ratio of compatible IM_PAIRs per local IM_PAIR and a large number of local IM_PAIRs take more CPU resources during the peer matching process. For example, as shown in Figure 15a, for 140 objects with 12 methods each, the latency was around 2000 microseconds per peer matching.

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 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. Figures 16a and 16b show these results respectively. Both figures also show that there is no noticeable performance penalty for supporting both client-server and context-sensitive communications with respect to processing outgoing data.

Experiment 4: This experiment also corresponds to RKF's Phase 3. However, unlike in Experiment 3, here we are interested in measuring RKF's incoming object-data processing latency. Similar to Experiment 3, we designed two different versions. The first version measured the latency when the data were received using context-sensitive communication channels. Figure 17a 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 17b.

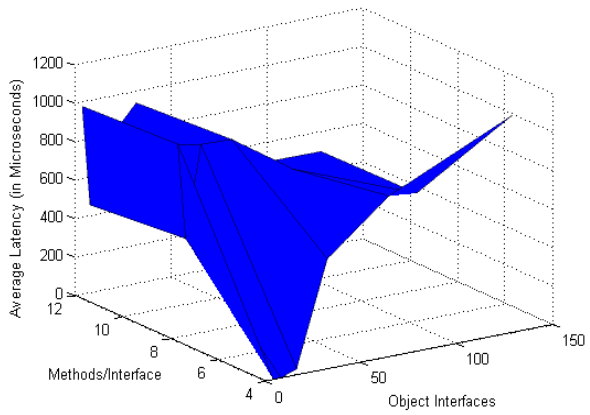
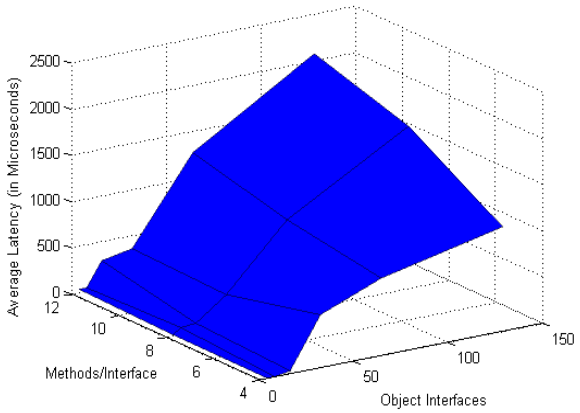
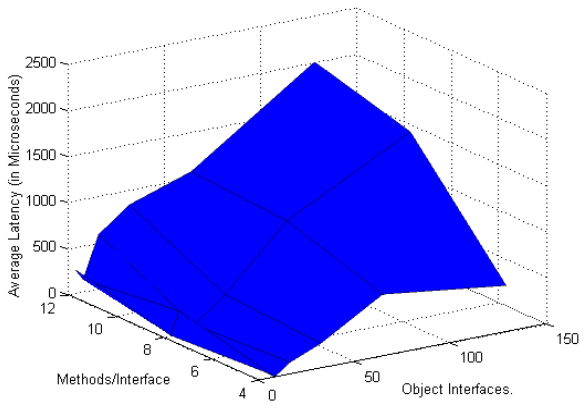


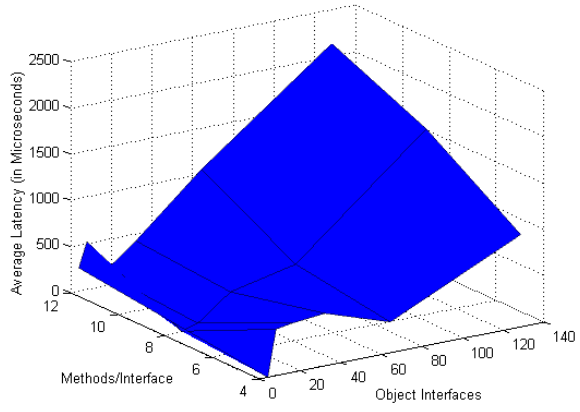
Figure 14: Context-match event processing latency in RKF during context-sensitive object information broadcast.



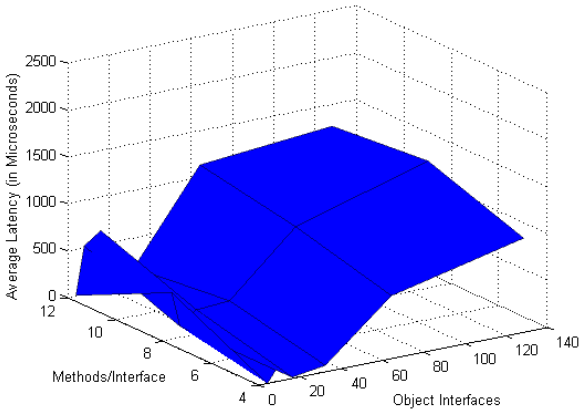
(a)



(b)



(c)



(d)

Figure 15: Peer matching latency in RKF for FRIEND_LIST/IM_PAIR = (a) 16, (b) 12, (c) 8 and (d) 4.

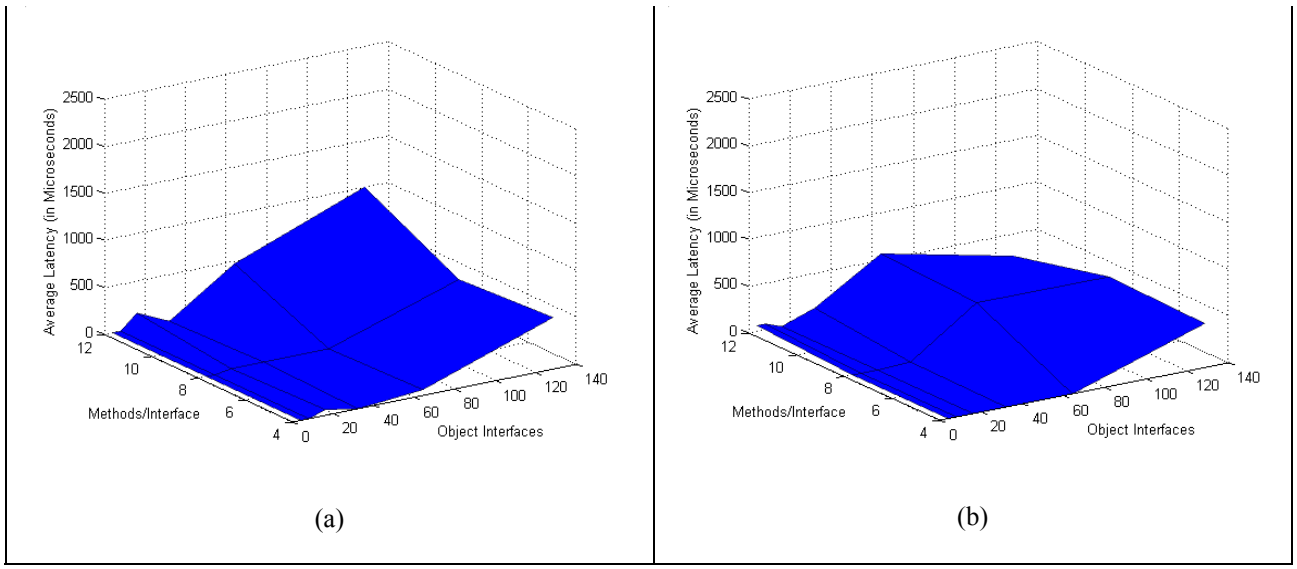


Figure 16: Outgoing object data processing latency in RKF during (a) context-sensitive communications, and (b) client-server communications.

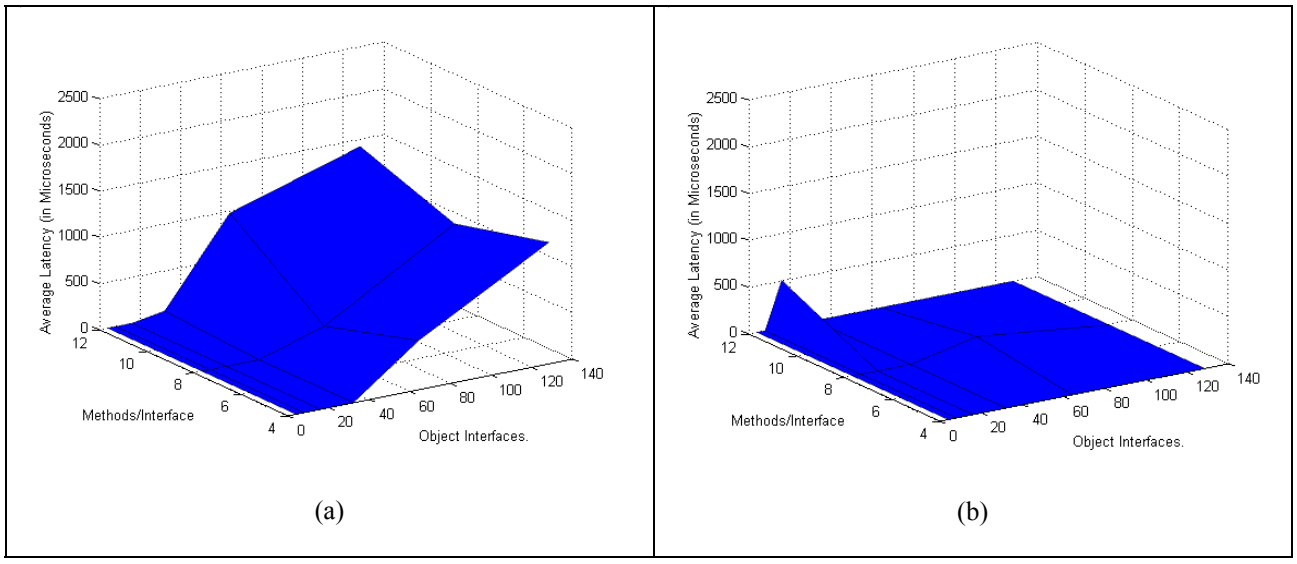


Figure 17: Incoming object data processing latency in RKF during (a) context-sensitive communications, and (b) client-server communications.

Lessons Learned from the Experiments: Based on these experimental results, it is noted that RKF performs a little more efficiently in case of client-server communications. The added complexity of context-sensitive communications adds to the slight increase in the latency in Figures 9, 10a-10d, 11a, and 12a. However, this added latency is negligible since even in worst cases (with more than 100 objects with 10 more methods each) the latency is in milliseconds. One important note is that since RKF operates in mobile ad hoc environments, the increased probability of packet collisions and loss in wireless environments may reduce the end-to-end performance of RKF.

- RCSM is lightweight, not only based on its implementation in Windows CE-based devices, but also its focus only on the core services of a middleware. In Windows CE-based Casio PDAs, R-ORB is only 50 KB large. A recent porting of R-ORB to Windows XP-based desktop showed that it only consumes 250 KB of space. This leads us to conclude that RCSM can be used to develop lightweight middleware services.
- RCSM's context acquisition and propagation mechanism is efficient. The complexity of this process is O (n). The main performance bottleneck can occur due to context propagation, if the total size of the data needed to represent context attributes increase substantially. In addition, R-ORB often needs to use device drivers to collect data from various types of sensors. Since device drivers are handled by the operating systems, unpredictable timing may result in some cases. This leads us to conclude that for extremely time-constrained devices, it is better to leave out the context acquisition and propagation functionality outside the ORB.
- Context acquisition and propagation mechanism is, on average, 10 times as expensive as RKF's operation. This leads us to conclude that context acquisition and propagation can be a performance bottleneck for other operations of an ORB if not considered carefully.
- RCSM can be used to work in different sensitivity levels, which mainly depend on how soon objects wish to detect changes in the environments. However, higher sensitivity requires more processing resources. This leads us to believe that a system analysis model needs to be developed that could analyze the objects' sensitivity levels as a specific Quality of Service (QoS) attribute. In addition, such a model should provide meaningful tradeoff analysis with other crosscutting requirements of the system.
- RKF operates very efficiently. Most algorithms of RKF have O(n) complexities. RKF, however, performs a little better for client-server objects than for context-sensitive objects, and its performance mainly depends on the number of ready-to-be-activated object-interfaces at any instant.

- RKF is a proactive protocol. It works by assuming that other devices are always interested in the local ready-to-be-activated objects. The protocol may cause other "uninterested" devices to drain battery power unnecessarily whenever a mobile ad hoc network becomes stationary or only a few devices in a network are interested in discovering new objects. Nevertheless, its asymmetric object advertisement scheme (it only advertises incoming IM_PAIRs) is very useful to conserve energy. We are developing an adaptive version of RKF to address the energy efficiency aspect in RCSM.

11. Discussion

We have presented RCSM, a context-sensitive middleware for addressing the dynamic integration of mobile devices with network infrastructures. Specifically, we have discussed its RKF protocol, which transparently provides the context-awareness support so that mobile devices and network infrastructures can discover each other. In addition, we have discussed RCSM's dynamic resource discovery and distributed peer-matching capabilities. Our experimental results indicate that dynamic integration can be done efficiently in PDA-like devices with reasonably high-performance. Future work includes the extension of RKF to provide situation-awareness support for DI and the development of application-specific services, such as group communication, which can be used by applications following a successful dynamic integration. We are also working on the security aspect by adding an authentication protocol that is automatically triggered whenever the surrounding context of a mobile device changes. Finally, we are developing an energy-efficient version of RKF for performing DI with minimized energy consumption during the peer discovery process.

Acknowledgment

This research is supported in part by National Science Foundation under grant numbers 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, and Dazhi Huang and Yisheng Yao for many helpful discussions.

References

[1] J. Schiller, *Mobile Communications*, 1st edition, Addison-Wesley, 2000.
[2] I. Foster and C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure", Morgan Kaufman, 1st ed., 1998.
[3] A. K Dey, "Understanding and Using Context," *J. Personal and Ubiquitous Computing*, vol. 5, no. 1, Springer-Verlag, Berlin, Germany, Feb. 2001, pp. 4-7.
[4] M. Weiser, "The Computer for the Twenty-First Century", *Scientific American,* pp. 94-104, September 1991.
[5] B. Schilit, N. Adams, and R. Want, "Context-Aware Computing Applications", *Proc. IEEE Workshop on Mobile Computing Systems and Applications*, California, Santa Cruz, USA, 1994, pp. 85-90.
[6] N. Marmasse and C. Schmandt, "Location-aware Information Delivery with comMotion", *Proc. 2nd Int'l Symp. Handheld and Ubiquitous Computing (HUC 2K),* http://www.media.mit.edu/~nmarmas/cmHUC2k.pdf.

[7] 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.
[8] 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.
[9] 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.
[10] D. Garlan, et al., "Project Aura: Toward Distraction-Free Pervasive Computing," *IEEE Pervasive Computing*, vol. 1, no. 2, Apr.–June 2002, pp. 22–31.
[11] T. Kindberg and A. Fox, "System Software for Ubiquitous Computing," *IEEE Pervasive Computing*, vol. 1, no. 1, Jan.–Mar. 2002, pp. 70–81.
[12] M. L. Dertouzos, "The Future of Computing", *Scientific American*, vol. 281, no. 2, Aug. 1999, Scientific American, Inc., New York, pp. 52-55.
[13] G. Abowd and E. D. Mynatt, "Charting Past, Present, and Future Research in Ubiquitous Computing," *ACM Trans. on Computer Human Interaction*, vol. 7, no. 1, Mar. 2000, pp. 29–58.
[14] N. Sawhney and C. Schmandt, "Nomadic Radio: Speech and Audio Interaction for Contextual Messaging in Nomadic Environments," *ACM Trans. on Computer Human Interaction*, vol. 7, no. 3, Sept. 2000, pp. 353–383.
[15] S. D. Gribble, et al., "The Ninja Architecture for Robust Internet-Scale Systems and Services", Computer Networks, vol. 35, no. 4, 2001, pp. 473-497.
[16] B. Brumit, B. Meyers, J. Krumm, A. Kern, and S. Shafer, "EasyLiving: Technologies for Intelligent Environments", *Proc. 2nd Int'l Symp. Handheld and Ubiquitous Computing*, Sept 2000, pp. 12-27.
[17] S. S. Yau, F. Karim, Y. Wang, B. Wang, and S. Gupta, "Reconfigurable Context-Sensitive Middleware for Pervasive Computing," *IEEE Pervasive Computing,* 1(3), July-September 2002, pp. 33-40.
[18] 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*.
[19] S. S. Yau, Y. Wang, and F. Karim, "Development of Situation-Aware Application Software in Ubiquitous Computing Environments", *Proc. 26th IEEE Int'l Computer Software and Applications Conference (COMPSAC 2002)*, August 2002, Oxford, UK, pp. 233-238.
[20] 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.
[21] A. Murphy, G. Picco, and G.-C. Roman, "LIME: A Middleware for Physical and Logical Mobility", *Proc. 21st Int'l Conf. Distributed Computing Systems*, Phoenix, USA, April 2001, pp. 524-533.
[22] IBM Research, TSpaces Project, http://www.almaden.ibm.com/cs/TSpaces/.
[23] IBM Research, Bluedrekar Project, http://www.research.ibm.com/BlueDrekar/.
[24] A. T. Campbell, M. E. Kounavis, R. R.-F., Liao, and O. and Angin, "The Mobiware Toolkit: Programmable Support for Adaptive Mobile Networking. *IEEE Personal Communications*. Paper location: http://comet.ctr.columbia.edu/mobiware/.
[25] 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, vol. 220.
[26] 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*, Germany. May 2001, http://www.cs.ucl.ac.uk/staff/L.Capra/publications.html.
[27] P. A. Bernstein, "Middleware: A Model for Distributed Services", *Communications of the ACM*, 39(2), February 1996, pp. 86-97.
[28] Object Management Group, "A Discussion of the Object Management Architecture", January1997, http://www.omg.org/technology/documents/formal/object_management_architecture.htm.
[29] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1st edition, 1995.