In Proceedings of the 1st International Conference on Embedded and Ubiquitous Computing, Aizu, Japan, August 2004. Copyright Springer Verlag.

The 3DMA Middleware for Mobile Applications*

Tore Fjellheim[1], Stephen Milliner[1], Marlon Dumas[1], and Kim Elms[2]

[1] Centre for Information Technology Innovation,
Queensland University of Technology
(t.fjellheim,s.milliner,m.dumas)@qut.edu.au
[2] SAP Research Centre, Brisbane
kim.elms@sap.com

Abstract. Mobile devices have received much research interest in recent years. Mobility raises new issues such as more dynamic context, limited computing resources, and frequent disconnections. These handle these issues, we propose a middleware, called 3DMA, which introduces three requirements, 1) distribution, 2) decoupling and 3) decomposition. 3DMA uses a space based[3] middleware approach combined with a set of "workers" which are able to act on the users behalf either to reduce load on the mobile device, or to support disconnected behavior. In order to demonstrate aspects of the middleware architecture we consider the development of a commonly used mobile application.

1 Introduction

In recent years there has been an increased interest in the use of mobile devices, both in the research area, and by industry. Analysis shows that by 2006 there will be over 760 million mobile users connected to the Internet and over 1.7 billion users by 2007 [8]. However, limitations of mobile devices pose many challenges for application programmers. Limited memory space and slow CPUs impose restrictions on locally running programs. Introducing more memory and faster CPU, however, requires more power and hence limit battery life. Connection characteristics on mobile devices vary from high bandwidth W-LAN connections to low bandwidth GSM connections. In either case frequent disconnections may occur, either unexpectedly (for example due to low battery) or voluntarily by a user, resulting in intermittent or completely severed connections. Allowing programs to continue operating under such variable circumstances is problematic. In addition, the active communications itself drains battery power. Further complicating the issues of mobility, is the fact that mobile users and devices change context frequently (e.g. location, bandwidth, activity). Context is defined by Dey [7] as any information which can be used to characterize the situation of an entity. To handle the limitations of mobile devices as well as changing context, it is crucial to provide support for context. A change in context may affect user

* This research is funded in part by SAP Research Centre, Brisbane

[3] Space based: A combination of tuple space, JavaSpace and blackboard architectural paradigms

requirements or preferences, and different users may react differently to changes in context. This points to user centric computing, or personalized computing as an important aspect which our architecture also seeks to support.

This project aims to combine previous independent solutions into a coherent middleware architecture to handle device limitations, and support context awareness and user centric computing. This has lead us to the design of the 3DMA(Decoupled, Distributed, Decomposed Mobile Applications) architecture. The middleware based on the 3DMA architecture aims to provide this support through the implementation of the notion of a "worker". A worker allows for applications and user knowledge to be pushed remotely, to be used for adaptation, coordination, and automatic remote processing during disconnections.

This paper will first detail requirements for mobility used in design of the 3DMA architecture (Section 2). Then the workers, and how they address the issues in mobility are described (Section 3). The 3DMA middleware will then be outlined (Section 4), followed by a case study (Section 5) with performance experiments (Section 6), which shows how 3DMA can be used to cope with mobility issues.

2 Architectural Requirements: The 3 D's

In this section we outline the three architectural requirements: Decomposition, Distribution and Decoupling.

Decomposition Decomposition supports existing techniques used to address limitations in mobile computing. For example, offloading can be used to address limitations on the mobile device, and benefits from a decomposed application, so applications do not have to be migrated in their entirety. Remote execution is another technique which increases the processing capabilities and execution environment of a device. Remote execution requires that the functionality is decomposed into components and placed on remote powerful servers. A third technique is functionality adaptation[1], which means changing the way processing is done, for example by swapping a component. This can be used to change functionality according to context. Support for decomposition is required for functionality adaptation to enable replacement of only small parts of the functionality. In 3DMA we assume support for decomposition by having the program built using components. Individual components can then be offloaded or replaced.

Distribution Distribution can alleviate resource constraints [17], and also support disconnected operation[26]. Disconnection is supported either by loading all required functionality locally before a disconnection occurs [25] or, by offloading local functionality and allowing remote processing to continue without user interaction. Both of these requires distribution support. Distribution can be done before, or at runtime. Because pre-runtime distribution strategies do not take into account the dynamic nature of mobile environments, especially with disconnections in mind, we take a dynamic distribution strategy. Distribution has

the risk of increasing battery usage, as well as response time. Increased battery usage may in turn lead to premature disconnections. Distribution also makes a program susceptible to disconnection, by requiring remote access to access some components. A system which supports distribution of components, should therefore also support disconnection.

Decoupling Decoupling allows entities to exist independently of each other, making it easier to change, replace or migrate individual components or services. Decoupling also facilitates asynchronous communication and message buffering, which has been used to address disconnected behavior in mobile systems [4]. Communication decoupling has proven useful in mobile environments due to the dynamic variation in connectivity characteristics. Communication decoupling can be in both time and space as defined by [12]. Time decoupling removes the need for processes to exist simultaneously, which facilitates disconnections. Space decoupling means that two communicating processes do not need to have knowledge of each other or exist on the same server, which makes the system flexible. Tuple spaces are good for storing and sharing information between many users [18], which may be required in context aware systems[27]. Furthermore, decoupling allows users to send a message, disconnect, reconnect later and check for replies. Decoupling therefore supports the notion of disconnected operation.

3 3DMA Detailed Architecture

To aid in overcoming limitations, and to support context awareness and user centric processing, 3DMA incorporates the notion of workers. There are four types of workers defined: 1) triggers, 2) connectors, 3) advisors and 4) context facets. These are detailed below.

Workers are used to coordinate or execute processing and are able to handle various types of requests, and participate in performing processing to the best interest of the user. Workers embody programmer and user knowledge of an application, and can be defined and created at or prior to runtime. Workers do not provide any services themselves, but specify for other components how processing is to be performed. Workers may use the knowledge they are given without requiring user interaction. By placing a worker remotely, user actions can be accomplished without further CPU/memory or communication cost, thereby increasing the benefits of distribution. Another aim of the workers is to allow context aware reactions to take place remotely.

Triggers. A trigger is defined as an element which reacts to specific events. Events could for example be the change of a user's location, bandwidth dropping below a certain threshold, or a service returning a result. A trigger can be used to connect services, by forwarding requests for processing to the correct services, and sending replies from one service, to another service. Triggers can also be used to automate processing when disconnected. Such automated processing could be initiated by context change or other events, such as the termination of a remote process.

Advisors. Advisors maintain preferences and policies of users. They are created through a mapping from a user preferences model to a set of advisors. Advisors are used to perform user centric processing through preferences, as well as to aid in resolving data or functionality heterogeneity. For example by selecting which functionality to upload to the mobile device.

Context Facets Context facets are the storage point for context information regarding one context element. They are similar to context widgets used by the context toolkit [22]. An entity may have several context facets such as location, traveling pace or bandwidth associated with it. Context Facets facilitate automatic processing during disconnections, as well as other adaptation which requires context awareness.

Connectors Connectors are used to connect devices to the board. There is one connector per device. Each connector is the server side representation of this remote device or computer. Connectors read messages from the board and forward them to the correct device. The connector will know the capabilities of its device, such as previously loaded functionality and bandwidth (known through a context facet). The connector will use this knowledge to make delivery decisions. Connectors support logical mobility to deliver functionality to capable devices, and thereby supports distribution.

4 The 3DMA Middleware Infrastructure

The 3DMA middleware infrastructure consists of three main elements (See Figure 1). The mobile device (MD), the JavaBoard(www.javaboard.org) object-space service(JB), and the execution environments (EE).

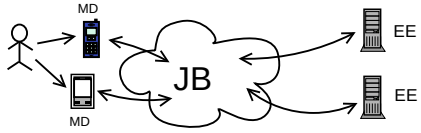

Fig. 1. System Overview

4.1 The JavaBoard

This section discusses the space based approach and the JavaBoard, with background, and reasons for our choice. The space based approach combines aspects of tuple spaces[12], JavaSpaces[11] and blackboard architectures[9].

A tuple space is a distributed shared memory space used for decoupled communication via sending and receiving messages. A tuple is an ordered set of

typed fields [3]. Processes can read and write tuples to the board. JavaSpaces expands the notion of tuples by using objects instad, creating an object-space.

The JavaBoard uses the tuple space communication methods to achieve decoupling. It supports the publish/subscribe pardigm which has prooven useful for disconnected operation in mobile environments[2]. It uses objects as in JavaSpaces and implements both state and code migration, thereby supporting the distribution requirement. Both tuple spaces and JavaSpaces however lack the notion of "Active Objects" (AO). Active Objects are supported in the JavaBoard, and are objects which execute when written to the space. Active Objects can be read and written to the board just as standard objects, which makes them easy to distribute. Active Objects are similar to "experts" used in Blackboard architectures.

Blackboard architectures provide a shared memory space in which problems are offered, and several participants (experts) can contribute towards a solution [9]. These experts are similar to the 3DMA notion of workers. The workers exist in the space, and communicate with each other and other entities through passing of messages via the space. The workers can be created dynamically by the applications to support personalized processing and sometimes automatic processing for individual users.

4.2 Execution Environments (EE) and Mobile Device (MD)

The EEs provide various capabilities to the mobile devices. These capabilities may be remote processing support, by picking up offloaded components, or they may be specific services such as document conversion. An EE can run as an AO or as a client separate from the board. They subscribe to certain type of objects depending on which service it offers. An EE may for example subscribe to requests for processing, or to offloaded components. If an EE picks up an offloaded component it will subscribe to processing directed at that component.

The mobile device (MD) can be viewed as a limited version of an EE. By using the JB and the EE, storage for offloading, increased processing, and specialized functionalities otherwise not available on the mobile device is provided. Using workers, the mobile device can also coordinate processing between requested services and thereby limit its communications requirement.

5 Case Study: Calendar Manager

The aim of the case study is to show some of the main benefits of our approach using a calendar manager scenario. The benefits include support for disconnected operation and distribution of functionality. We also show the advantages of the workers and a decoupled architecture. The following sections will describe some issues which can arise in a calendar manager setting, and explain how they would be handled in our architecture. The issues discussed are:

- Heterogeneity: Varying calendar data
- Multi-Channel: Access to same calendar from many terminals
- Planned Disconnections: Multi user task coordination

5.1 Issue 1: Heterogeneity

Heterogeneity is a problem in mobile environments which exists due to multiple technologies both in hardware and software. We show two approaches to handling this issue. Firstly by calling an external service to translate the data, and secondly by delivering new functionality to process the new data.

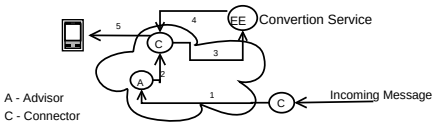

Fig. 2. Issue 1 - Solving the heterogeneity problem using a translator.

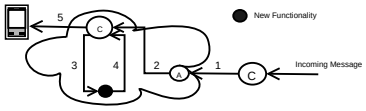

Fig. 3. Solving the heterogeneity problem using dynamic delivery.

Approach 1: Calling External Services This issue illustrates how the system interacts with external services. An example service could be to adapt data to changing context such as low battery or variable bandwidth. This adaptation can be specified and performed during runtime, thereby allowing end users to specify their adaptation requirements. By using advisors the system can vary which service is used depending on context. Remote adaptation without device knowledge can thereby be done, thereby saving battery, processing and bandwidth. Also, new services can also be added dynamically. If data specific adaptation is required a data element could be augmented with data type, to support type-specific translation (e.g. images, text, video etc.). An advisor would then forward the data to the correct conversion service.

Figure 2 shows how 3DMA can handle heterogeneity issues through a translation service. The service subscribes to processing requests from the JavaBoard. The connector will subscribe to data to be sent to the device. If the connector detects that data is in the wrong format (e.g. wrong type, or too large), a request for processing is done by placing a request for conversion on the board. The translator will then process the request.

Approach 2: Dynamic Delivery An alternative solution to some heterogeneity problems is dynamic delivery of required functionality, currently not loaded on the mobile device. Dynamic delivery of functionality has been attempted, as a way to keep functionality local. We use it to solve the heterogeneity problem by delivering the functionality required to process the message. Which component that is chosen will depend on issues such as the current device context and user preferences. This is shown in figure 3. A message is put on the board. It is picked up by an advisor which sends it to the correct connector. The connector will know that the required functionality is not available on the mobile device, locates the required functionality, and delivers it with the message.

Uploading of new functionality for a calendar manager could for example be used to plan a meeting. One user could arrange a meeting time and could place some functionality (on the JavaBoard) which other users can use to sign up for the meeting. Interested parties would load the functionality and either sign up for it, or suggest alternative times. The functionality is then used to set up a collaborative environment. After the time had been arranged the user planning the meeting could upload the functionality to book a room at a specific location (e.g. hotel).

5.2 Issue 2: Multi Channel

Multi-channel delivery means that information can be delivered through one or several available channels such as direct socket communication, SMS or e-mail. This could be required if the mobile device is not currently linked to the network, if the calendar application is currently shut down or if the user wants to have the same calendar accessible from many terminals. In addition it may be desirable to limit the amount of traffic to the mobile device, and have some communications redirected, or the users context may even dictate that notifications about certain calendar events are sent to the users current terminal.

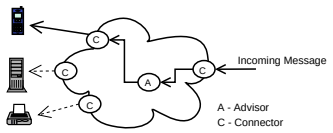

Fig. 4. Issue 2 - Multiple Channels. The advisor accepts a incoming message to this user, and forwards it to the correct connector.

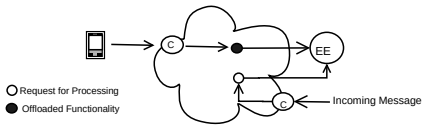

Fig. 5. Issue 3 - Disconnected Multi User Sharing. The offloaded component in the execution environment recieves all incoming messages until reconnection.

Multi channel delivery is handled as shown in Figure 4. A request is received by the incoming connector, and an object is placed on the board. This object is then picked up by the connector(s) listening for this type of message. The connectors allow delivery decisions to be made before actual delivery to the device. If there was a direct coupling, the data would be sent to the terminal immediately. By using connectors and advisors all terminals may subscribe to the same data, but they can communicate before delivery to agree on what to deliver and to which channel it should be delivered.

In case of disconnection, an advisor could be used to specify a redirection to another connector, or the message would be stored on the board until reconnection. In case of the advisor, a subscription to a context facet is required to know when disconnection has occured. The advisor would pick up the request instead of the connector, and would change the direction of it to the correct

connector. It is also possible to specify a trigger to react to an event signaling that a connector is disconnected. This could in turn call upon a connector to attempt a reconnection.

5.3 Issue 3: Planned Disconnection

If users share the same calendar, they may like to recieve notification on other users events and plans. If a user is disconnected, notification on information can still occur by migrating a component to an execution environment prior to disconnection, as shown in figure 5. All incoming messages to this component will then be directed to the execution environment instead of the device. The offloaded component is then able to process these messages.

This approach allows users to be disconnected while still allowing components which are remote to receive information. It may also be possible for other users to query the users calendar while the user is disconnected. This facilitates saving of battery, communication, and processing on the local device, which may be beneficial even when a connection is available.

6 Preliminary Performance Evaluation: Task Sequencing

Preliminary performance validation of the architecture was done to discover how the 3DMA architecture compares in terms of remote execution response time, to other standard systems such as sockets and RMI. A calendar manager could use remote execution for many reasons. For example to call on offloaded components, or utilize external services.

One of the advantages our architecture is to be able to build new services by combining existing services. We therefore show how the 3DMA architecture can be used to increase performance over heterogeneous links when utilizing two or more services in sequence.

In all tests, the JavaBoard service was running on a desktop PC. A HP i740e with built in W-LAN was used as the mobile device. Communications from the mobile device was done over a 11Mb WLAN connection. The PC also ran the service(s) connected to the JavaBoard. Initial performance tests compared 3DMA to socket communication and RMI. Our tests showed that performance of these protocols are approximately equal and they outperform 3DMA by 250-300 ms. Also, response time increases almost linearly as message sizes increase. The results confirm that the JavaBoard is slower because of the extra processing required. We do however, not consider this to be a significant performance overhead. This is because the 3DMA architecture have other advantages such as ease of sharing data and functionality, resilience to disconnections as well as automatic remote processing. This last element can be used to combine services to achieve better response time. Several remote execution requests are thereby done in sequence, by only sending one request. Figure 6 shows an example a calendar manager might use to execute multiple services using 3DMA triggers.

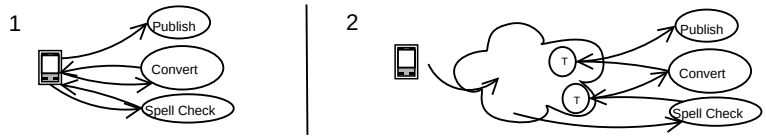

Fig. 6. Frame 1 shows a standard multiple service invocation from the mobile device. Three message are sent (document is sent three times), and two received. Frame 2 shows how 3DMA would do multiple service invocations, by defining two triggers and then send a request to execute the first service. Three messages are sent (document and two triggers), and none received.

Sequencing of tasks using triggers can be used to reduce response time, and to reduce battery usage by reducing the amount of data transferred form the mobile device. In a traditional scenario as shown in figure 6, should a disconnection occur after the first request has been sent, the processing would be further delayed by waiting for the user to reconnect. Our approach allows the user to disconnect earlier.

The response time results of a double service invocation is shown in figure 7. The size of the transmittet trigger was 1Kb. This reduced the amount of communication from the mobile device. As the size of messages grows, 3DMA outperforms sockets. This indicates that the network is the bottleneck. Should the connection be even slower (e.g. GSM/GPRS), or the number of services increase the benefits would increase even more [16].

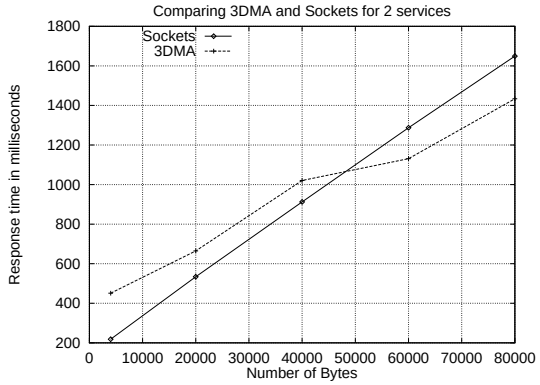

Fig. 7. 3DMA vs Sockets

7 Related Work

Decomposition in previous mobile computing research has for example been used to automatically partition the runtime Java class tree into two parts, to alleviate

memory and CPU constraints[13]. Other projects did static decomposition [17], and aimed to reduce battery consumption through distribution. Systems such as J-Orchestra [24] and Coign [14] create components which later can be distributed dynamically. 3DMA takes a static component approach to decomposition but supports dynamic distribution.

Decoupled communication has been widely explored. In Lime [20] the Linda [12] tuple space was expanded for mobile environments. Lime supports code mobility and bulk operations, to retrieve all matching tuples in a read or take. Further extensions have been made in the Limbo system [5], for example by implementing priorities for important messages and using system agents. The TOTA system [18] provides a tuple space based system for context aware pervasive applications, where there is no centralized located space, but tuples are rather inserted into the network. TOTA aims to provide a space in which context data can easily be shared between many entities. The JavaBoard space goes beyond other space systems by using Active Objects. This, for example, provides more support for offloading and disconnected behaviour. A similar notion to Active Objects is "Agent Wills"[21], which operate in tuple spaces to provide fault tolerance. Active Objects can be used to implement such Agent Wills, but also provide a more general framework.

In context awareness, Schilit [23] was one of the first to propose a system architecture. Later, but significant contributions to the evolution of infrastructures for context awareness are the Context Toolkit [22], and [27]. The latter providing a space based approach. Our approach to context awareness is a hybrid between the Context Toolkit and the space based approach.

There are many approaches to disconnected operation. In the FarGo-Da system [26] and in JAMP [25] disconnected operation is supported by loading components up locally onto the mobile device when a disconnection is imminent. However this approach fails when disconnections are unexpected. The Coda file system [15] attempts to keep document copies local in case of a disconnection. The XMiddle system [19] and the Bayou architecture [6] attempts to share information between users in an environment of frequent disconnections. These projects consider data sharing, and conflict resolution when updates occur, whereas 3DMA consider functionality distribution. Both are required to properly support disconnection in a mobile environment. Another approach to disconnections was taken by Kottmann et.al. [16], in which a series of remote calls could be called simultaneously and asynchronously thereby requiring less connectivity. This was done by using components called delegators and trustees. 3DMA workers builds further on the potential of these components.

8 Conclusion and Future Work

In the future it would be desirable to have a way to specify and develop applications in this environment. One aim is to develop an API to support decoupling, distribution and decomposition, so programmers can easily incorporate workers into their application. Also having a formal specification of how the workers in-

teract would be desirable, to be able to fully understand the behavior and detect if the system is lacking. A component model based on the FarGo[26] structure is also being developed.

Other experiments such as possible reduction in battery power through dynamic delivery, saving memory or CPU by substituting components, or lowering communication usage through offloading are being considered. Similar experiments have previously been addressed in other environments [10, 13]. Scalability by using multiple JavaBoards is also being planned.

Mobile environments requires an architecture which will help alleviate the limited resources on mobile devices, support context awareness, and allow user centric processing. The 3DMA architecture aims to support all these issues. Decoupling provides the flexibility required for a mobile computation system. Decomposition in turn allows more fine grained adaptation to take place, and distribution helps limit resource constraints, and supports disconnection. These three principles hold the key to an efficient mobile middleware system. The provided 3DMA middleware, has been proven through implementation and evaluation of several issues which may occur in a real application, as well as through performance testing.

References

1. N. M. Belaramani, C. Wang, and F. C. M. Lau. Dynamic component composition for functionality adaption in pervasive environments. In *Proceedings of The 9th IEEE Workshop on Future Trends of Distributed Computing Systems*, San Juan, Puerto Rico, May 2003.
2. I. Burcea, H.A. Jacobsen, E. de Lara, V. Muthusamy, and Petrovicm M. Disconnected operation in publish/subscribe middleware. In *Proceedings of the 2004 IEEE International Conference on Mobile Data Management (MDM'04)*, 2004.
3. A. Corradi, F. Zambonelli, and L. Leonardi. A scalable tuple space model for structured parallel programming. In *Proceedings of the Conference on Massively Parallel Programming Models*, 1995.
4. G. Cugola, E. Di Nitto, and G. P. Pico. Content-based dispatching in a mobile environment. In *Proceedings of The Workshop on Distributed Systems: Algorithms Architectures and Languages*, September 2000.
5. N. Davies, S. P. Wade, A. Friday, and G.S. Blair. Limbo: A tuple space based platform for adaptive mobile applications. In *Proceedings of The 23rd International Conference on Open Distributed Processing/Distributed Platforms*, 1997.
6. A. Demers, K. Petersen, M. Spreitzer, D. Terry, M. Theimer, and B. Welch. The bayou architecture: Support for data sharing among mobile users. In *Proceedings of IEEE Workshop on Mobile Computing Systems & Applications*, 1994.
7. A. K. Dey and G. D. Abowd. Towards a better understanding of context and context-awareness. In *Proceedings of the 2000 Conference on Human Factors in Computing Systems*, The Hague, The Netherlands, April 2000.
8. P. Drews, D. Sommer, R. Chandler, and T. Smith. Managed runtime environments for next-generation mobile devices. *Intel Technology Journal*, 7(1), 2003.
9. L. D. Erman, F. Hayes-Roth, and R. D. Reddy. The hersay-ii speech understanding system: Integrating knowledge to resolve uncertainty. *ACM Transactions on Programming*, 12(2), 1980.

10. J. Flinn, S. Park, and M. Satyanarayanan. Balancing performance, energy and quality in pervasive computing. In *Proceedings of The 22rd International Conference on Distributed Computing*, 2002.
11. E. Freeman, S. Hupfer, and K. Arnold. *JavaSpaces: principles, patterns, and practice*. Addison-Wesley, Boston, Massachusetts, 1999.
12. D. Gelernter. Generative communication in linda. *ACM Transactions on Programming*, 2(1):80–112, January 1985.
13. X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic. Adaptive offloading inference for delivering applications in pervasive computing environments. In *Proceedings of The 1st International Conference on Pervasive Computing and Communications*, Fort Worth, Texas, March 2003.
14. G.C. Hunt and M.L. Scott. The coign automatic distributed partitioning system. In *Proceedings of the 3rd symposium on Operating System Design and Implementation*, Febuary 1999.
15. J.J. Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. In *Proceedings of The Thirteenth ACM Symposium on Operating Systems Principles*, 1992.
16. D. Kottmann, R. Wittmann, and M. Posur. Delegating remote operation execution in a mobile computing environment. *Mobile Networks and Applications*, 1(4), 1996.
17. Z. Li, C. Wang, and R. Xu. Computation offloading to save energy on handheld devices: A partition scheme. In *Proceedings of The International conference on Compilers, architecture, and synthesis for embedded systems*, November 2001.
18. M. Mamei, F. Zambonelli, and L. Leonardi. Programming context-aware pervasive computing applications with tota, 2002.
19. C. Mascolo, L. Capra, S. Zachariadis, and W. Emmerich. Xmiddle: A data-sharing middleware for mobile computing. *Int. Journal on Personal and Wireless Communications.*, April 2002.
20. G. P. Picco, A. L. Murphy, and G.-C. Roman. Lime: Linda meets mobility. In *Proceedings of The 21st International Conference on Software Engineering*, Los Angeles, California, May 1999.
21. A. Rowstron. Using mobile code to provide fault tolerance in tuple space based coordination languages. *Science of Computer Programming*, 46, 2003.
22. D. Salber, A. K. Dey, and G. D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *Proceedings of The ACM SIGCHI Conference on Human Factors in Computing Systems*, May 1999.
23. W. N. Schilit. *A System Architecture for Context-Aware Mobile Computing*. PhD thesis, Columbia University, 1995.
24. E. Tilevich and Y Smaragdakis. J-orchestra: Automatic java application partitioning. In *Proceedings of the 16th European Conference on Object Oriented Programming*, june 2002.
25. M. Valente, R. Bigonha, M. Bigonha, and A. Loureiro. Disconnected operation in a mobile computation system. In *Proceedings of ICSE*, 2001.
26. Y. Weinsberg and I. Ben-Shaul. A programming model and system support for disconnected-aware applications on resource-constrained devices. In *Proceedings of the 24th international conference on Software engineering*, pages 374–384. ACM Press, 2002.
27. T. Winograd. Architectures for context. *Human-Computer Interaction*, 16(2,3 and 4), 2001.