Programming Language Support for Adaptable Wearable Computing *

P. K. McKinley, S. M. Sadjadi, E. P. Kasten and R. Kalaskar

Software Engineering and Network Systems Laboratory
Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan 48824
{mckinley,sadjadis,kasten,kalaskar}@cse.msu.edu

Abstract

This paper investigates the use of programming language constructs to realize adaptive behavior in support of collaboration among users of wearable and handheld computers. A prototype language, Adaptive Java, contains primitives that permit programs to modify their own operation in a principled manner. In a case study, Adaptive Java was used to construct MetaSocket components, whose composition and behavior can be adapted to changing conditions during execution. MetaSockets were then integrated into Pavilion, a web-based collaboration framework, and experiments were conducted on a mobile computing testbed containing wearable, handheld, and laptop computer systems. Performance results demonstrate the utility of MetaSockets to improving the quality of interactive audio streams and reliable data transfers among collaborating users.

Keywords: adaptive middleware, reflection, wearable computing, mobile computing, wireless networks, forward error correction.

1 Introduction

The large-scale deployment of wireless communication services and advances in wearable computers and other mobile devices, enable users to collaborate in new ways. Example applications include computer-supported cooperative work, collaborative scientific experimentation, management of industrial installations, and command and control systems. However, given their synchronous and interactive nature, collaborative applications are particularly sensitive to the heterogeneous characteristics of both the computing devices and the network connections used by participants.

Wearable computers, in particular, pose several challenges to the design of collaborative applications. An application must tolerate the highly dynamic channel conditions that arise as the user moves about the environment. Moreover, the application must accommodate devices with widely varying input devices and display characteristics. Finally, an application must tolerate limited system resources, relative to that of workstations, in terms of processor speed, memory size, and power. To enable effective collaboration among users in such environments, including transferring applications from one device to another, the systems must adapt to these conditions at run time.

Adaptability can be implemented in different parts of the system. One approach to is to introduce a layer of adaptive communication-oriented middleware between applications and underlying transport services [14, 24, 25, 39, 43]. An adaptive middleware framework can help to insulate application components from platform variations and changes in external conditions. On the other hand, many *context-aware* applications are more effective when they explicitly take advantage of the dynamics of the environment [8]. As a result, a number of supporting frameworks have been proposed to assist the application developer in managing context and adapting to changing conditions [9–11, 13, 23, 32, 38].

Regardless of what parts of the system implement adaptive behavior, an important issue is how to adapt to situations unforeseen during the original software development. This problem is especially important to systems that must continue to operate (correctly) during exceptional situations. Examples include systems used to manage critical infrastructures, such as power grids and nuclear facilities, as well as command and control environments. Such systems require run-time adaptation, including the ability to modify and replace components, in order to survive hardware com-

* This work was supported in part by the U.S. Department of the Navy, Office of Naval Research under Grant No. N00014-01-1-0744, and in part by National Science Foundation grants CDA-9617310, NCR-9706285, CCR-9912407, EIA-0000433, and EIA-0130724.

ponent failures, network outages, and security attacks.

We are currently conducting an ONR-sponsored project called *RAPIDware* that addresses the design of adaptive middleware services to support critical infrastructure protection in dynamic, heterogeneous environments. The RAPIDware project complements many of those cited earlier by focusing on software technologies and programming abstractions for building adaptable systems. The techniques use rigorous software engineering principles to help preserve functional properties of the code as the system adapts to its environment. As part of this study, we have developed Adaptive Java [18], an extension to Java that supports dynamic reconfiguration of software components.

In this paper, we use Adaptive Java to support dynamic reconfiguration of collaborative applications executed on wearable computers. The main contribution is to show by experimentation on a mobile computing testbed that these language constructs provide an effective way to implement adaptable components. We anticipate that the Adaptive Java language may be useful to other researcher groups that are investigating adaptability in ubiquitous computing environments. Section 2 provides an overview of our experimental environment, including background on Pavilion, a collaborative framework used in our study. Section 3 describes the Adaptive Java language. In Section 4, we describe the use of Adaptive Java to transform normal Java sockets into "metamorphic" sockets, or MetaSockets, which support run-time modifications to their operation and interfaces. Section 5 describes the use of MetaSockets to support adaptive error control on audio channels and reliable multicast in Pavilion, including results of a performance study on a mobile testbed that includes three Xybernaut MA-V wearable computers. Section 6 discusses related work, and Section 7 presents our conclusions and discusses future directions.

2 Experimental Environment

The RAPIDware project is largely experimental. All the software techniques we are developing are implemented and evaluated on a mobile computing testbed. The testbed includes various types of mobile computers: several 1Gz Dell laptop computers (bootable in either Windows 2000 or Linux), several Compaq iPAQ handheld systems (some runing Windows CE, others running Linux) and three Xybernaut Mobile Assistant V wearable computers (each with a 500 MHz processor and 256M memory). These systems currently communicate via an 11Mbps 802.11b wireless local area network (WLAN). Our local wireless cell is also connected to a a multi-cell WLAN that covers many areas of the MSU Engineering Building and its courtyard; see Figure 1.

To support our investigations of collaborative computing across heterogeneous environments, we previously devel-

Figure 1. Users of the mobile computing testbed in the courtyard of the MSU Engineering Building.

oped an object-oriented groupware framework called Pavilion [29]. Pavilion is written in Java and supports collaboration using off-the-shelf browsers such as Netscape Navigator and Microsoft Internet Explorer. In default mode, Pavilion operates as a collaborative web browser, as depicted in Figure 2. A member of the group acquires control of the session through the leadership protocol. On the leader's system, shown on the left in Figure 2, the browser interface monitors the activities of the web browser. The interface is notified whenever a new URL is loaded by the browser, and it reliably multicasts this URL to all other participants. The web resource itself and any embedded/linked files are reliably multicast by the leader's proxy server to the proxy servers of the other group members. At each receiving system, the browser interface requests the local web browser to load the new URL. The target web browser will subsequently initiate retrieval of the files, via its proxy, which will return the requested items. While browsing, the collaborating users can speak with each other through real-time audio channels [31]. In addition to supporting collaborative browsing, Pavilion components can be reused and extended in order to construct new collaborative applications. For example, Pavilion has been used to develop VGuide [5], a collaborative virtual reality application that enables a user to select any VRML file from the Internet and lead a group of users through that virtual world.

Pavilion was originally designed for wired network environments. We later extended Pavilion to wireless networks by constructing proxy servers to meet the needs of mobile computers [31]. Although these proxies support run-time adaptability, their adaptation techniques are *ad hoc*, rather than supported by the language (Java) or the run-time system. In the RAPIDware project, we seek principled approaches, based on programming abstractions and rigorous software engineering methods, to streamline the de-

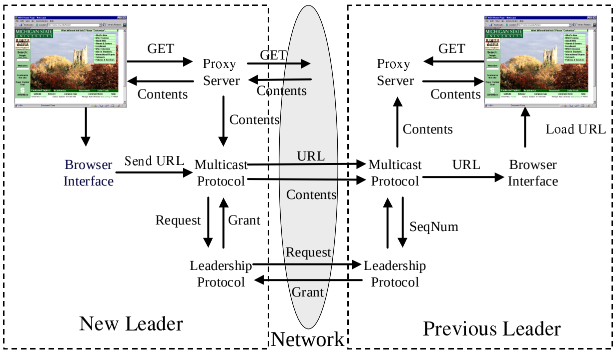

Figure 2. Default operation of Pavilion.

velopment and maintenance of distributed computing systems, while enhancing their capability for automatic self-configuration and adaptation. In the remainder of this paper, we describe Adaptive Java and how we used it to realize adaptability in the Pavilion framework when executed on wearable systems.

3 Adaptive Java Background

Adaptive Java [18] is based on computational reflection [26, 37], which refers to the ability of a computational process to reason about (and possibly alter) its own behavior. Typically, the *base-level* functionality of the program is augmented with one or more *meta* levels, each of which observes and manipulates the base level. In object-oriented environments, the entities at a meta level are called meta-objects, and the collection of interfaces provided by a set of meta-objects is called a meta-object protocol, or MOP.

The basic building blocks used in an Adaptive Java program are *components*, which can be thought of as adaptable classes. The key programming concept in Adaptive Java is to provide three separate component interfaces: one for performing normal imperative operations on the object (*computation*), one for observing internal behavior (*introspection*), and one for changing internal behavior (*intercession*). Operations in the computation dimension are referred to as *invocations*. Operations in the introspection dimension are called *refractions*: they offer a partial view of internal structure and behavior, but are not allowed to change the state or behavior of the component. Operations in the intercession dimension are called *transmutations*: they are used to modify the computational behavior of the component.

This separation of interfaces is intended to address a key issue that arises in the use of reflection, namely, the degree to which the system should be able to change its own behavior [19]. A completely open implementation implies that an application can be recomposed entirely at run-time, which may produce undesired behavior. On the other hand, limiting adaptability also limits the ability of the system to survive adverse situations. Hence, rather than considering MOPs as orthogonal portals into base-level functionality [7], we propose an alternative model in which MOPs are constructed from primitives, namely, refractions and transmutations. Different MOPs can be defined for different cross-cutting concerns: communication quality-of-service, fault tolerance, security, energy management, and so on. We argue that defining different MOPs in terms of a common set of primitives facilitates the coordination of their activities.

An existing Java class is converted into an adaptable component in two steps, as shown in Figure 3. First a *base-level* Adaptive Java component is constructed from the Java class through an operation called *absorption*, which uses the `absorbs` keyword. As part of the absorption procedure, mutable methods called *invocations* are created on the base-level component to expose the functionality of the absorbed class. Invocations are mutable in the sense that they can be added and removed from existing components at run-time using meta-level transmutations. We emphasize that the relationship between invocations on the base-level component and methods on the base-level class need not be one-to-one. Some of the base-level methods may be occluded or even combined under a single invocation as the system's form is modified. In this manner, the base-level component defines explicitly those parts of the original class are to be adaptable.

In the second step, *metafication* enables the creation of refractions and transmutations that operate on the base component, as shown in Figure 3. Meta components are defined using the `metafy` keyword. We emphasize that the meta-level can also be given a meta-level, which can be used to refract and transmute the meta-level. In theory, this reification of meta-levels for meta-levels could continue infinitely [26]. Refractions and transmutations embody limited adaptive logic and are intended for defining *how* the base level can be inspected and changed. The logic defining *why and when* these operations should be used is provided at other meta levels or by other components entirely.

We used CUP [15], a parser generator for Java, to implement Adaptive Java Version 1.0, which is used in this study. CUP takes our grammar productions for the Adaptive Java extensions and generates an LALR parser, called ajc, which performs a source-to-source conversion of Adaptive Java code into Java code. Semantic routines were added to this parser such that the generated Java code could then be compiled using a standard Java compiler.

4 MetaSocket Design and Operation

We used Adaptive Java to develop an adaptable component called a MetaSocket. By using MetaSockets instead of normal Java sockets, an application or middleware ser-

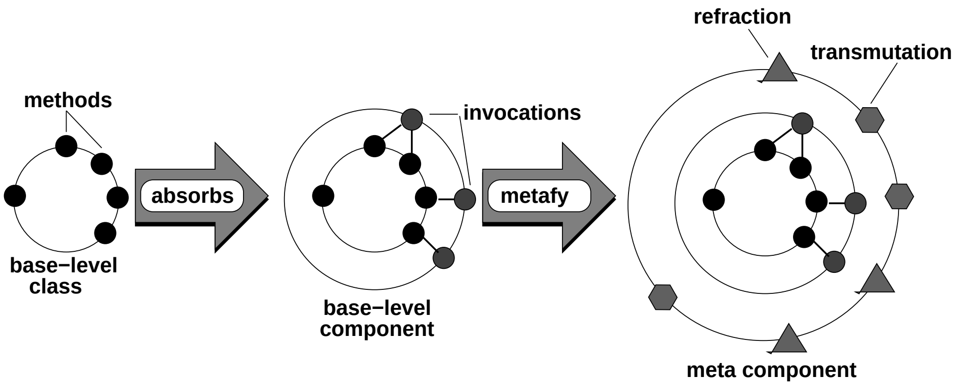

Figure 3. Component absorption and metafication.

vice can dynamically observe and change its behavior in response to external events. In the RAPIDware project, we are using MetaSockets for several purposes, including reporting traffic patterns for intrusion detection, reducing energy consumption when the battery is low, and to managing the quality-of-service of network connections. In the study described herein, we integrated MetaSockets into the Pavilion collaborative framework and experimented with adaptive error control for wearable computers interconnected by a wireless LAN.

The characteristics of wireless LANs are very different from those of their wired counterparts. Factors such as signal strength, interference, and antennae alignment produce dynamic and location-dependent packet loss [30]. These problems affect multicast connections more than unicast, since the 802.11b MAC layer does not provide link-level acknowledgements for multicast frames. Forward error correction (FEC) can be used to improve reliability by introducing redundancy into the data channel. As shown in Figure 4, an (n, k) *block erasure code* converts k source packets into n encoded packets, such that any k of the n encoded packets can be used to reconstruct the k source packets [35]. Hence, in multicast data streams, as used by collaborative applications, a single parity packet can be used to correct independent single-packet losses among different receivers.

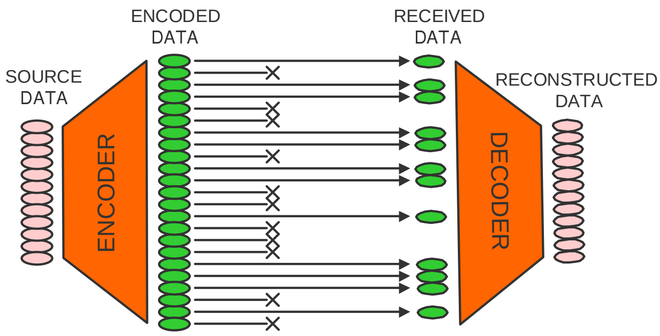

Figure 4. Operation of block erasure code.

Figure 5 depicts the structure of a `MetaSocket` component that has been configured with a two-filter pipeline. A *filter* is a piece of code that applies a particular computation, such as FEC or compression, to a data stream [31]. The base-level component, called `SendSocket`, was created by absorbing the existing Java `Socket` class. Certain public members and methods are made accessible through invocations on `SendSocket`. This particular instantiation is intended to be used only for sending data, so the only invocations available to other components are `send()` and `close()`. Hence, the application code using the computational interface of a metamorphic socket looks similar to code that uses a regular socket. The `SendSocket` was metafied to create a meta-level component called `MetaSocket`. `GetStatus()` is a refraction that is used to obtain the current configuration of filters. `InsertFilter()` and `RemoveFilter()` are transmutations that are used to modify the filter pipeline.

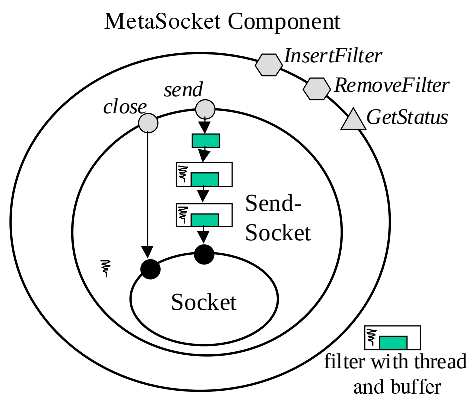

Figure 5. Structure of a MetaSocket.

Separate components, called a Decision Makers, reside within each application and control the behavior of adaptive components such as MetaSockets. For example, a Decision Maker might use refractions on a MetaSocket to monitor packet loss behavior and decide to modify the filter configuration using a transmutation. For purposes of testing the MetaSocket interfaces, we also developed an interactive administration utility that enables us to manipulate MetaSockets directly.

5 Experimental Results

The availability of an adaptable socket-like component enabled us to construct an adaptable version of Pavilion for use in wearable computers and other mobile devices. Specifically, we replaced the Java sockets with MetaSockets and added decision maker components to adjust their behavior at run time. Next, using the wearable computers in our testbed, we experimented with the MetaSockets used for interactive audio streaming among participants and for reliable multicasting of web resources.

Interactive Audio Streaming. First, we investigated the use of MetaSockets to enhance the quality of wireless audio channels at run time. The audio streaming code comprises two main parts. On the sending station, the Recorder uses the `javax.sound` package to read audio data from a system's microphone and multicast it on the network. On the receiving station, the Player receives the audio data and plays it using `javax.sound`. The audio encoding uses a single channel with 8-bit samples. Relatively small packets are necessary for delivering audio data, in order to reduce jitter and minimize losses [31]. Hence, each packet contains 128 bytes, or 16 msec of audio.

We experimented with the transmutative interface to MetaSockets by dynamically inserting and removing FEC filters from the MetaSockets on the sending and receiving sides of the audio stream. In these tests, we used our interactive GUI, instead of an autonomous decision maker, to manipulate the metasockets. We streamed audio across the wireless LAN from a 1GHz laptop computer to a Xybernaut MA-V system and dynamically inserted FEC filters in the respective MetaSockets. Figure 6 shows five superimposed traces, corresponding to the insertion of FEC filters with $k = 4$ and $n = 6, 8, 10, 12$, and 16, respectively. For each trace, the user of the wearable computer moved farther away from the wireless access point, generally producing a lower signal-to-noise ratio and higher packet losses. The x axis shows number of packets being sent, each unit representing a set of 200 data packets (or 50 4-packet FEC groups), and the y axis shows the percentage of data packets lost. In all traces, an FEC filter is inserted at packet set 20 and removed at packet set 40. As shown, the filters are very effective in reducing the packet loss.

Reliable Multicasting. The delivery of web resources among Pavilion programs is provided by the Web-Based

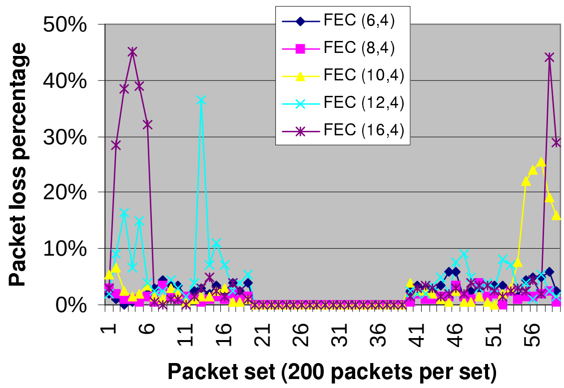

Figure 6. Dynamic FEC on audio MetaSockets.

Reliable Multicast (WBRM) protocol [28], an application-level protocol that implements reliability atop UDP/IP multicast. Referring back to Figure 2, The WBRM protocol is a receiver-initiated, or NAK-based, protocol: a receiver notifies the sender only when it misses a packet in the stream [41]. Figure 7 shows the WBRM protocol architecture. Both the sending and receiving components of the protocol comprise a set of Java threads and data structures. The sender maintains a log vector describing the resource stream and a cache of recently transmitted resources. The receiver sends NAKs for missing packets, while buffering those that arrive intact but out-of-order.

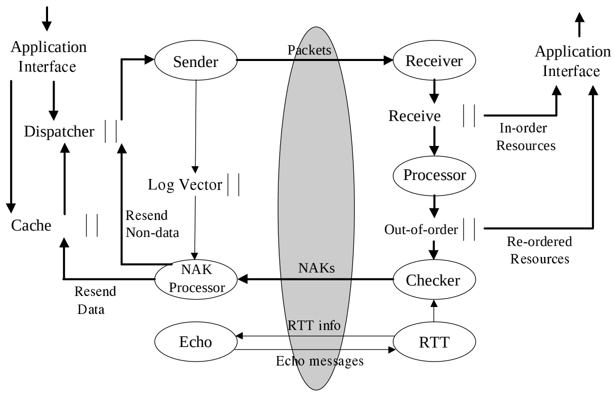

Figure 7. WBRM protocol architecture.

In this study, we again insert FEC filters into MetaSockets at run time. Figure 8 shows typical results near the periphery of the wireless cell without FEC. The dark vertical lines indicates packet losses that result in NAKs and retransmissions, whereas light lines indicate successful packet delivery). Using a simple (6,4) FEC filter, the delivery rate increases dramatically.

Figure 9 shows the latency results, with and without the FEC filter, for downloading different sized files from a wireless laptop to one of the MA-V wearable systems near the

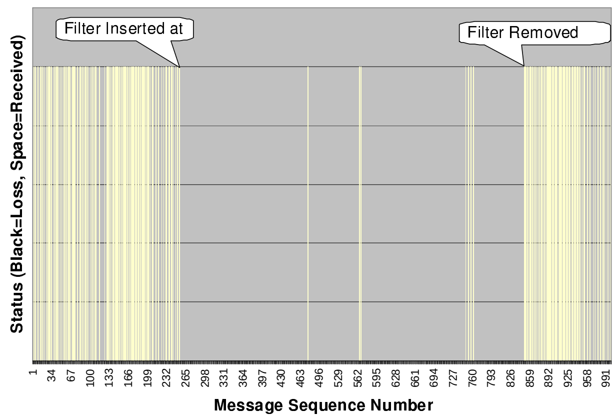

Figure 8. Trace of packet losses in WBRM.

periphery of the wireless cell. The results show the average of 5 trial runs. The reduction in latency ranges from 18% for a 50Kbyte file to 36% for a 1Mbyte file. We emphasize that, given the high error rate at this location, even FEC cannot produce a very high throughput. The theoretical limit of 802.11b is 7 Mbps [17], and a highly tuned C++ program can achieve over 6 Mbps [40]. The Java WBRM protocol can achieve about 4 Mbps when the receiver is near the access point. However, the performance using MetaSockets in this remote location is comparable to what we can achieve with a tuned Java proxy server. We report only initial results here, and we are continuing our investigations. Moreover, the use of MetaSockets (and Adaptive Java, in general) facilitates a cleaner separation between adaptive code and application code. Indeed, we did not touch the base Pavilion or WBRM code in these tests.

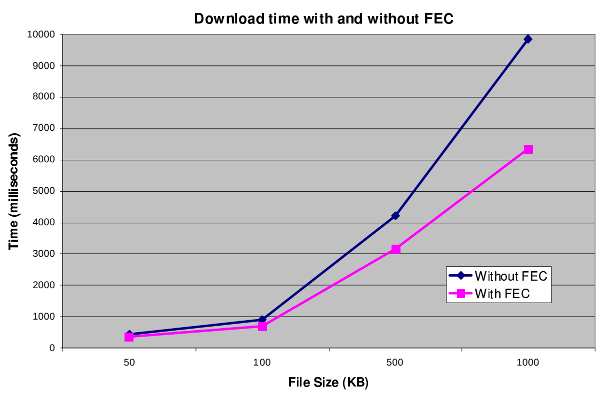

Figure 9. Reliable multicast latency.

6 Related Work

In recent years, numerous research groups have addressed the issue of adaptive middleware frameworks that can accommodate dynamic, heterogeneous infrastructures. Examples include Adapt [12], MOST [14], Rover [16], MASH [27], TAO [24], dynamicTAO [22], MobiWare [4], MCF [25], QuO [43], MPA [36], Odyssey [34], DaCapo++ [39], RCSM [44], and Sync [33]. In addition, several higher-level frameworks have been designed to support wearable/ubiquitous applications; examples include Hive [32], Ektara [9], and Proem [23], Puppeteer [13], Aura [38], and the Context Toolkit [10].

These and related projects have greatly improved the understanding of how a system can adapt to changes in the environment and in user behavior and interactions. Our work in the RAPIDware project complements such contributions by focusing on principled approaches to adaptive software design that include programming language support and rigorous software engineering methods. Such support holds the promise that compile-time and run-time checks can be performed on the adaptive code in order to help ensure consistency and preservation of certain key properties as the system changes. Moreover, these techniques facilitate the run-time adaptation of the system in ways not anticipated during the original development.

Other researchers have addressed the use of programming language constructs to realize adaptable behavior. For example, Andersson and Ritzau [3] describe a method to support dynamic update of Java programs, but that technique requires a modified JVM. Our "weaving" of adaptive code with the base application is reminiscent of aspect-oriented programming [21]. Although many projects in the AOP community focus on compile-time weaving [20], a growing number of projects focus on run-time composition [2, 42]. By defining a reflection-based component model, Adaptive Java also supports run-time reconfiguration but is not restricted to the AOP model that requires identification of predefined "pointcuts" at compile time. A related concept is composition filters [6], which provide a mechanism for disentangling the cross-cutting concerns of a software system. Besides filters, however, Adaptive Java can be applied to components that interact in arbitrary ways, and therefore is perhaps more general.

The PCL project [1] also focuses on language support for run-time adaptability and is perhaps most closely related to our work. PCL is intended for use directly by applications. Our concept of "wrapping" classes with base components is similar to the use of *Adaptors* used in PCL. However, modification of the base class in PCL appears to be limited to changing variable values, whereas Adaptive Java transmutations can modify arbitrary structures or subcomponents. Moreover, by combining encapsulation with metafication, Adaptive Java can be used to realize adaptations in multiple meta-levels.

7 Conclusions and Future Directions

In this study, we investigated the application of Adaptive Java to support run time adaptation in wearable computers. We demonstrated the use of MetaSockets in extending Pavilion, a collaborative computing framework, to mobile wearable systems. In particular, we used the run-time transmutative capability of MetaSockets to improve their resiliency to packet losses on a wireless LAN. While the examples in this paper are both communication services, we emphasize that the Adaptive Java mechanisms are general. Currently, we are conducting several subprojects where we are using Adaptive Java to address other key areas where software adaptability is needed in wearable computers and other mobile devices: dynamically changing the fault tolerance properties of components; adaptive security policies dynamically woven across components; mitigation of the heterogeneity of system display characteristics; and energy management strategies for battery-powered devices.

Further Information. A number of related papers and technical reports of the Software Engineering and Network Systems Laboratory can be found at the following URL: `http://www.cse.msu.edu/sens`.

References

[1] V. Adve, V. V. Lam, and B. Ensink. Language and compiler support for adaptive distributed applications. In *Proceedings of the ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (OM 2001)*, Snowbird, Utah, June 2001.

[2] F. Akkai, A. Bader, and T. Elrad. Dynamic weaving for building reconfigurable software systems. In *Proceedings of OOPSLA 2001 Workshop on Advanced Separation of Concerns in Object-Oriented Systems*, Tampa Bay, Florida, October 2001.

[3] J. Andersson and T. Ritzau. Dynamic code update in JDrums. In *Proceedings of the ICSE'00 Workshop on Software Engineering for Wearable and Pervasive Computing*, Limerick, Ireland, 2000.

[4] O. Angin, A. T. Campbell, M. E. Kounavis, and R.R.-F.M. Liao. The Mobiware toolkit: Programmable support for adaptive mobile networking. *IEEE Personal Communications Magazine, Special Issue on Adapting to Network and Client Variability*, August 1998.

[5] J. Arango and P. K. McKinley. VGuide: Design and performance evaluation of a synchronous collaborative virtual reality application. In *Proceedings of the IEEE International Conference on Multimedia and Expo*, New York, July 2000.

[6] L. Bergmans and M. Aksit. Composing crosscutting concerns using composition filters. *Communications of the ACM*, 44(10):51–57, October 2001.

[7] G. S. Blair, G. Coulson, P. Robin, and M. Papathomas. An architecture for next generation middleware. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98)*, The Lake District, England, September 1998.

[8] G. Chen and D. Kotz. A survey of context-aware mobile computing research. Technical Report TR2000-381, Computer Science Department, Dartmouth College, Hanover, New Hampshire, November 2000.

[9] R. W. DeVaul and A. Pentland. The Ektara architecture: The right framework for context-aware wearable and ubiquitous computing applications. The Media Laboratory, Massachusetts Institute of Technology, unpublished, 2000.

[10] A. K. Dey and G. D. Abowd. The Context Toolkit: Aiding the development of context-aware applications. In *Proceedings of the Workshop on Software Engineering for Wearable and Pervasive Computing*, Limerick, Ireland, June 2000.

[11] S. Fickas, G. Kortuem, and Z. Segall. Software organization for dynamic and adaptable wearable systems. In *Proceedings First International Symposium on Wearable Computers (ISWC'97)*, Cambridge, Massachusetts, October 1997.

[12] T. Fitzpatrick, G. Blair, G. Coulson, N. Davies, and P. Robin. A software architecture for adaptive distributed multimedia applications. *IEE Proceedings - Software*, 145(5):163–171, 1998.

[13] J. Flinn, E. de Lara, M. Satyanarayanan, D. S. Wallach, and W. Zwaenepoel. Reducing the energy usage of office applications. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, pages 252–272, Heidelberg, Germany, November 2001.

[14] A. Friday, N. Davies, G. Blair, and K. Cheverst. Developing adaptive applications: The MOST experience. *Journal of Integrated Computer-Aided Engineering*, 6(2):143–157, 1999.

[15] S. E. Hudson, editor. *CUP User's Manual*. Usability Center, Georgia Institute of Technology, july 1999.

[16] A. D. Joseph, J. A. Tauber, and M. F. Kaashoek. Mobile computing with the Rover toolkit. *IEEE Transactions on Computers: Special issue on Mobile Computing*, 46(3), March 1997.

[17] A. Kamerman and G. Aben. Net throughput with ieee 802.11 wireless LANs. In *Proceedings of the IEEE Wireless Communications and Networking Confernce 2000 (WCNC 2000)*, volume 2, pages 747–752, 2000.

[18] E. Kasten, P. K. McKinley, S. Sadjadi, and R. Stirewalt. Separating introspection and intercession in metamorphic distributed systems. In *Proceedings of the IEEE Workshop on Aspect-Oriented Programming for Distributed Computing (with ICDCS'02)*, Vienna, Austia, July 2002. to appear.

[19] G. Kiczales. Towards a new model of abstraction in the engineering of software. In *International Workshop on Reflection and Meta-Level Architecture*, Tama-City, Tokyo, Japan, nov 1992.

[20] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An overview of aspectj. In *ECOOP*, pages 327–353, 2001.

[21] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Videira Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*. Springer-Verlag LNCS 1241, June 1997.

[22] F. Kon, M. Román, P. Liu, J. Mao, T. Yamane, L. C. M. aes, and R. H. Campbell. Monitoring, security, and dynamic configuration with the dynamicTAO reflective ORB. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2000)*, New York, April 2000.

[23] G. Kortuem, J. Schneider, D. Preuitt, T. G. C. Thompson, S. Fickas, and Z. Segall. When peer-to-peer comes face-to-face: Collaborative peer-to-peer computing in mobile ad-hoc networks. In *Proceedings of the 2001 International Conference on Peer-to-Peer Computing (P2P2001)*, Linköpings, Sweden, August 2001.

[24] F. Kuhns, C. O'Ryan, D. C. Schmidt, O. Othman, and J. Parsons. The design and performance of a pluggable protocols framework for object request broker middleware. In *Proceedings of the IFIP Sixth International Workshop on Protocols For High-Speed Networks (PfHSN '99)*, Salem, Massachusetts, August 1998.

[25] B. Li and K. Nahrstedt. A control-based middleware framework for quality of service adaptations. *IEEE Journal of Selected Areas in Communications*, 17(9), September 1999.

[26] P. Maes. Concepts and experiments in computational reflection. In *Proceedings of the ACM Conference on Object-Oriented Languages (OOPSLA)*, dec 1987.

[27] S. McCanne, E. Brewer, R. Katz, L. Rowe, E. Amir, Y. Chawathe, A. Coopersmith, K. Mayer-Patel, S. Raman, A. Schuett, D. Simpson, A. Swan, T. Tung, D. Wu, and B. Smith. Toward a common infrastructure for multimedia-networking middleware. In *Proc. 7th Intl. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '97), St. Louis, Missouri*, May 1997.

[28] P. K. McKinley, R. R. Barrios, and A. M. Malenfant. Design and performance evaluation of a Java-based multicast browser tool. In *Proceedings of the 19th International Conference on Distributed Computing Systems*, pages 314–322, Austin, Texas, 1999.

[29] P. K. McKinley, A. M. Malenfant, and J. M. Arango. Pavilion: A distributed middleware framework for collaborative web-based applications. In *Proceedings of the ACM SIGGROUP Conference on Supporting Group Work*, pages 179–188, November 1999.

[30] P. K. McKinley and A. P. Mani. An experimental study of adaptive forward error correction for wireless collaborative computing. In *Proceedings of the IEEE 2001 Symposium on Applications and the Internet (SAINT-01)*, San Diego-Mission Valley, California, January 2001.

[31] P. K. McKinley, U. I. Padmanabhan, and N. Ancha. Experiments in composing proxy audio services for mobile users. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, pages 99–120, Heidelberg, Germany, November 2001.

[32] N. Minar, M. Gray, O. Roup, R. Krikorian, and P. Maes. Hive: Distributed agents for networking things. In *Proceedings of ASA/MA'99, the First International Symposium on Agent Systems and Applications and Third International Symposium on Mobile Agents*, 1999.

[33] J. Munson and P. Dewan. Sync: A system for mobile collaborative applications. *IEEE Computer*, 30(6):59–66, 1997.

[34] B. D. Noble and M. Satyanarayanan. Experience with adaptive mobile applications in Odyssey. *Mobile Networks and Applications*, 4:245–254, 1999.

[35] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM Computer Communication Review*, April 1997.

[36] M. Roussopoulos, P. Maniatis, E. Swierk, K. Lai, G. Appenzeller, and M. Baker. Person-level routing in the mobile people architecture. In *Proceedings of the 1999 USENIX Symposium on Internet Technologies and Systems*, Boulder, Colorado, October 1999.

[37] B. C. Smith. Reflection and semantics in Lisp. In *Proceedings of 11th ACM Symposium on Principles of Programming Languages*, pages 23–35, 1984.

[38] J. P. Sousa and D. Garlan. Aura: an architectural framework for user mobility in ubiquitous computing environments. In *Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture*, Montreal, Canada, August 2000. to appear.

[39] B. Stiller, C. Class, M. Waldvogel, G. Caronni, and D. Bauer. A flexible middleware for multimedia communication: Design implementation, and experience. *IEEE Journal of Selected Areas in Communications*, 17(9):1580–1598, September 1999.

[40] C. Tang. Adaptive reliable multicast in wireless local area networks. Master's thesis, Department of Computer Science and Engineering, Michigan State University, East Lansing, Michigan, 2002.

[41] D. Towsley, J. Kurose, and S. Pingali. A comparison of sender-initiated and receiver-initiated reliable multicast protocols. *IEEE Journal on Selected Areas in Communications 15, 3*, pages 398–406, April 1997.

[42] E. Truyen, B. N. Jörgensen, W. Joosen, and P. Verbaeten. Aspects for run-time component integration. In *Proceedings of the ECOOP 2000 Workshop on Aspects and Dimensions of Concerns*, Sophia Antipolis and Cannes, France, 2000.

[43] R. Vanegas, J. A. Zinky, J. P. Loyall, D. A. Karr, R. E. Schantz, and D. E. Bakken. QuO's runtime support for quality of service in distributed objects. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98)*, The Lake District, England, September 1998.

[44] S. S. Yau and F. Karim. Adaptive middleware for ubiquitous computing environments. In *Proceedings of IFIP WCC 2002 Stream 7 on Distributed and Parallel Embedded Systems (DIPES 2002)*, Montreal, Canada, August 2002. to appear.