Realizing Multi-Dimensional Software Adaptation

P. K. McKinley, E. P. Kasten, S. M. Sadjadi, and Z. Zhou

Software Engineering and Network Systems Laboratory
Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan 48824

{mckinley,kasten,sadjadis,zhouzhin}@cse.msu.edu

ABSTRACT

This paper describes the use of programming language constructs to support run-time software adaptation. A prototype language, Adaptive Java, contains primitives that permit programs to modify their own operation in a principled manner. In case studies, Adaptive Java is being used to support adaptation for different cross-cutting concerns associated with heterogeneous mobile computing and critical infrastructure protection. Examples are described in which Adaptive Java components support dynamic quality-of-service on wireless networks, run-time energy management for handheld computers, and self-auditing of potential security threats in distributed environments.

1. INTRODUCTION

Given today's heterogeneous mobile computing infrastructure, many distributed computing applications need to adapt to their environment in multiple ways. For example, communication software must accommodate wireless networks that are far less reliable and stable than their wired counterparts. In addition, user interfaces must conform to devices with widely varying display characteristics and capabilities, from conventional workstations to palmtop devices. Moreover, many systems must implement energy management services, given the reliance of mobile devices on battery power. Finally, applications must confront the vulnerability of a connectionless packet infrastructure by protecting themselves against intrusions and other security threats.

Adaptability can be implemented in different parts of the system. One popular approach introduces a layer of adaptive middleware between applications and underlying transport services, for example, see [11, 13, 23, 24, 43]. An appropriate middleware platform can help to insulate application components from platform and network variability and can simplify the implementation of fault tolerance and security services. Many approaches to adaptive middleware design, and adaptive software design in general, involve computational reflection [25, 38], which refers to the ability of a computational process to reason about (and possibly alter) its own behavior. A key issue that arises in the use of reflection, and one of the major issues addressed in our work, is the degree to which the system should be able to change its structure and operation. 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.

Our interest in this problem arises from our work on RAPIDware, an ONR-sponsored research project that addresses the design and use of adaptive middleware for protection of critical infrastructures, such as power grids, nuclear facilities, and command and control networks. The RAPIDware project focuses on developing unified software technologies, based on rigorous software engineering principles, to support different dimensions of adaptability while preserving functional properties of the code. Although we primarily target interactive collaborative applications, the techniques are general and could be applied to other domains, such as high-performance distributed computing or operation of peer-to-peer networks.

In an earlier paper [18], we proposed a new model for adaptive components is based on the concept of providing separate component interfaces for observing behavior (*introspection*) and for changing behavior (*intercession*). This separation is intended to simplify the development of adaptive functionality by restricting the ways in which components can be manipulated, thereby helping to ensure correctness and consistency. To further explore this model, we implemented a programming language called Adaptive Java, which supports dynamic reconfiguration of software components. In this paper, we survey our activities in using Adaptive Java to provide adaptable component behavior in three different cross-cutting concerns: communication quality-of-service, energy management, and security.

The remainder of the paper is organized as follows. In Section 2, we provide background on Adaptive Java, a particular adaptable component called the MetaSocket, and the mobile computing testbed on which we conducted our experiments. Section 3 describes the use of MetaSockets to support adaptable error control, for both interactive audio streaming and reliable multicasting, on wireless networks. Section 4 illustrates how adaptive communcation services can be used to manage the energy consumption of small handheld devices. Section 5 shows how Adaptive Java can be used to support adaptable auditing of security threats: effectively, any component of an application can be turned into an “informer” at run time. In each section, experimental results are presented. Section 6 discusses related work, and Section 7 presents our conclusions.

2. BACKGROUND

2.1 Adaptive Java.

The Adaptive Java language [18] is based on computational reflection [25, 38]. In reflective systems, typically, the *base-level* functionality of the program is augmented with one or more *meta* levels,

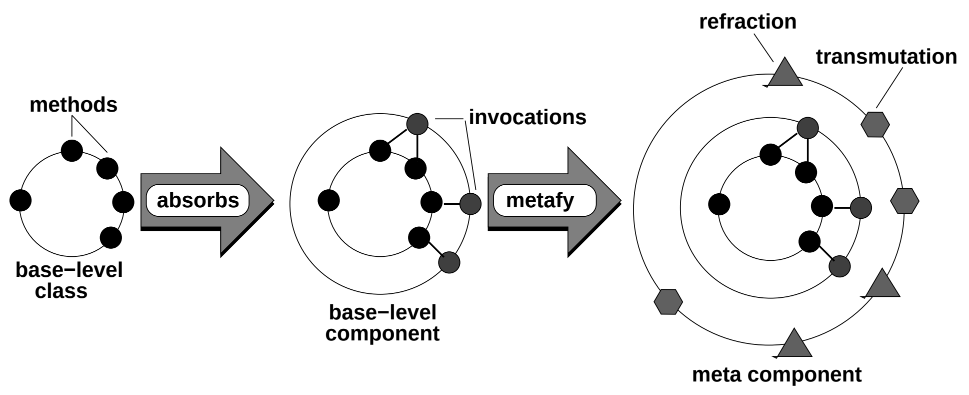


Figure 1: Component absorption and metafication.

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. 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, such as *decision makers*.

An existing Java class is converted into an adaptable component in two steps, as shown in Figure 1. 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. For example, we might create a base-level socket by absorbing a socket class. However, the base-level socket may provide a customized interface for use in a particular application domain.

In the second step, *metafication* enables the creation of refractions and transmutations that operate on the base component, as shown in Figure 1. 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 [25]. Continuing our socket example, a tranmutation might be defined to insert compression or encryption modules into a socket, while a refraction might be used to observe traffic patterns on behalf of an intrusion detection system.

Rather than considering MOPs as orthogonal portals into base-level functionality [8], we propose an alternative model in which MOPs are constructed from primitives, namely, refractions and transmutations. Figure 2 illustrates this concept. 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 through components such as decision makers and event mediators.

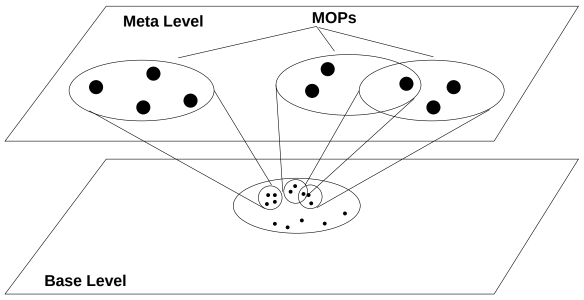


Figure 2: MOPs implemented with primitive operations.

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.

2.2 MetaSockets

Given the importance of communication services to distributed computing, the first component that we developed in Adaptive Java is the "metamorphic" socket, or simply, MetaSocket. Using MetaSockets instead of normal Java sockets enables that part of the application (or middleware service) to dynamically observe and change its behavior in response to external events. Figure 3 depicts the structure of a `MetaSocket` component that has been configured to perform two types of preprocessing, or *filtering*, on a data stream before it it actually sent using the internal Java socket.

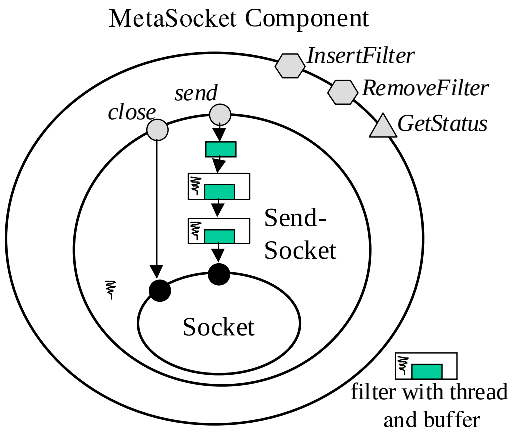


Figure 3: Structure of a MetaSocket component.

The base-level component, called `SendSocket`, was created by absorbing the an existing Java socket class. (in most of our experiments, we use the `MulticastSocket` class, but we ignore that detail here). 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. MetaSockets and other components can either change their own behavior or be acted upon by other components.

In the remainder of this extended abstract, we briefly discuss our experiments in adapting MetaSockets for different cross-cutting concerns; details will be provided in the full paper. In our study, we integrated MetaSockets into Pavilion [29], a collaborative computing application, and conducted experiments on a mobile computing testbed. The testbed includes various types of mobile computers: 1Gz Dell laptop computers, Compaq iPAQ handheld systems, and Xybernaut Mobile Assitant V wearable computers. These systems currently communicate via an 11Mbps 802.11b wireless local area network (WLAN), which is also connected to a a multi-cell WLAN that covers many areas of the MSU Engineering Building and a nearby courtyard; see Figure 3(b).

2.3 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 4.

Figure 4: Users of the mobile computing testbed in the courtyard of the MSU Engineering Building.

To support our investigations of collaborative computing across heterogeneous environments, we previously developed 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. 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 development 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 in heterogeneous wireless environments.

3. ADAPTABLE ERROR CONTROL

The characteristics of wireless networks 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 5, 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 [26]. These codes are particularly effective for multicast data streams, where a single parity packet can be used to correct independent single-packet losses among different receivers [36]. We have investigated the use of MetaSockets to alter dynamically the quality of two different types of multicast data streams: interactive audio and reliable multicasting of files. Both studies involve run-time insertion and adaptation of FEC filters in MetaSockets.

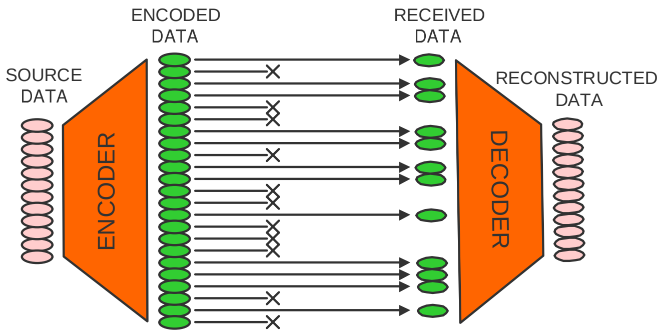


Figure 5: Operation of FEC based on block erasure codes.

In the audio streaming experiments, the audio is recorded on a wired workstation and transmitted to a wireless Xybernaut MA-V wearable computer. The application 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 the Java Sound API. Both applications were written in Adaptive Java and converted into pure Java using ajc. They communicate using MetaSockets instead of regular Java sockets.

The audio stream is transmitted over a 100 Mbps Ethernet LAN to a wireless access point, where it is multicast on an 11 Mbps 802.11b wireless LAN. 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 256 bytes, or 32 msec of audio. In the experiments, we used our interactive GUI, instead of an autonomous decision maker component, to manipulate the metasockets. Figure 6 shows five superimposed traces, where we streamed audio across the wireless LAN from a 1GHz laptop computer to a Xybernaut MA-V wearable computer. In all traces, an FEC filter is inserted at packet set 20 and removed it at packet set 40. As shown, the filters are very effective in reducing the packet loss.

In the case of reliable multicast, we replaced Java sockets with MetaSockets in WBRM [28], an application-level protocol that implements reliability atop UDP/IP multicast. In this study, we again insert FEC filters into MetaSockets at run time. Figure 7 shows typical results near the cell periphery without FEC (dark indicates packet loss, light indicates packet delivery). Using a simple (6,4) FEC filter, the delivery rate increases dramatically. Moreover, the performance using MetaSockets in this remote location is comparable to what we can achieve with a tuned Java proxy server. We report here only initial results here, and we are continuing our investigations. The use of MetaSockets (and Adaptive Java, in general) facilitates a cleaner separation of adaptive behavior from the rest of the application code.

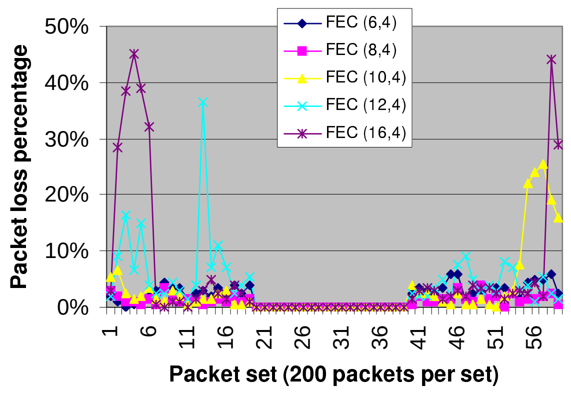


Figure 6: Dynamic insertion of FEC filter in audio stream.

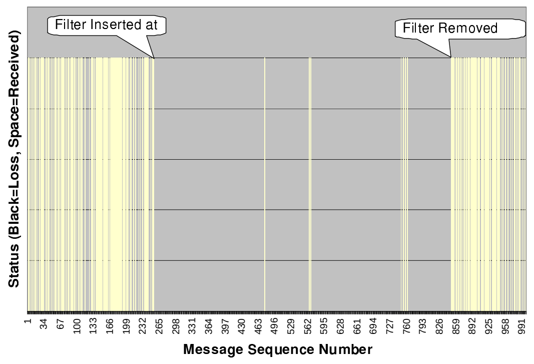


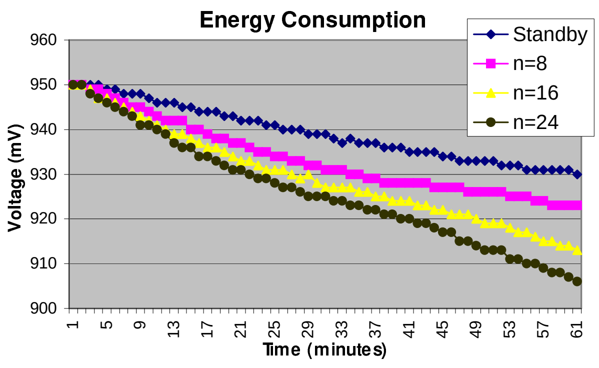
Figure 7: Reliable multicast trace.

4. ENERGY MANAGEMENT

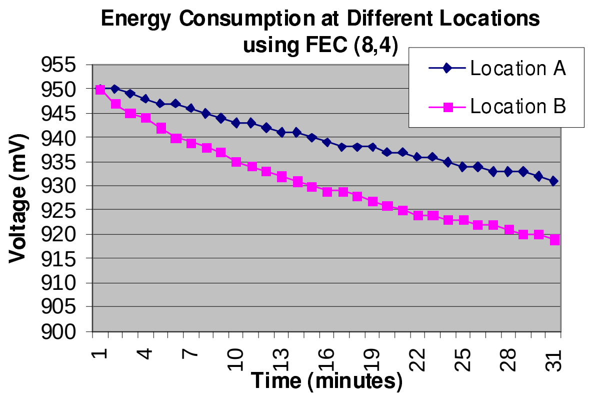
Until fundamental breakthroughs are made in providing power to handheld and wearable computers, battery lifetime will be a critical issue for such devices. In addition to error control and quality of service, the ability of the system to change the behavior of components at run time may also be useful in managing the total energy consumption of the device. To explore this hypothesis, we conducted a study in which we measured the effect on battery power in iPAQ H3670 handheld computers, each configured with a WLAN interface card. We again used MetaSockets, but we emphasize that any Adaptive Java component can potentially be adapted to address energy consumption issues.

The main iPAQ battery pack contains a Lithium Ion Polymer battery and a monitor chip that exams the battery behavior and status, such as voltage, temperature, chemistry, etc. We developed a simple application to record the voltage as the system is running. For reference in the following discussion, the measured voltage on a fully charged iPAQ is approximately 950 mV. On an otherwise idle system, the battery discharges linearly until it reaches approximately 850 mV, at which point the system will not operate (details of the second, external battery will be discussed in the full paper).

In our experiments, we used MetaSockets to receive different types of data streams, with FEC decoding performed as needed by filters configured in the MetaSockets. The iPAQ systems used in these tests run Windows CE, although we are currently conducting additional tests with Linux iPAQs. Figure 8(a) shows the results of varying FEC parameters and receiving a live audio stream. (The "Standby" curve is for an idle iPAQ equipped with an operational network interface card.) Different FEC parameters affect the energy consumption of the device due to (1) receiving of additional packets and (2) decoding of FEC groups. In Figure 8(a), the former is more important, since the the receiving iPAQ is only a short distance from the wireless access point, where packet losses are low. In Figure 8(b), we measured energy in two different locations. Location A is near to the access point, where the the signal strength is 100%, and Location B is outside the room containing the access point and approximately 10 meters down a hallway, where the signal strength is only 30%. The FEC parameters are fixed at (8,4). Although not shown, the packet loss rate *after* FEC decoding for both locations is approximately 5%. The difference in energy consumption is due to the additional invocations of the FEC decoder at Location B.



(a) different FEC parameters



(b) different locations

Figure 8: Energy consumption for different MetaSocket configurations.

Even these sample results indicate that the manner in which components are adapted has implications in terms of remaining battery life. However, saving battery life may produce undesired consequences in quality-of-service, security, and other cross-cutting aspects of the system. These trade-offs are the subject of our current studies. In the full paper, we will present additional results, including effects of different power saving modes in combination with different MetaSocket parameters.

5. SELF-AUDITING FOR SECURITY

Designing a highly trusted component-based software system requires that every component, in its turn, satisfies the security policy defined for the entire system. A good security framework needs to adapt to changing policies as well as respond to changing environmental conditions, including perceived threats to the system. A key part of any trusted system is the audit mechanism, which enables information to be collected for both off-line and on-line analysis. Traditionally, audit-related code is tangled with the base code of the application, implying that the security policy and requirements need to be known at development time. Recently, a number of improvements have been proposed, including the use of agent hierarchies [6], mobile agents [16, 35, 41], and compile-time weaving of security code into applications using aspect-oriented programming languages [44].

Adaptive Java provides an additional capability, namely, a means to insert security-related code into a components as it executes. By metafying a component with the appropriate set of refractions and transmutations, effectively, any component of an application can be turned into an "informer" at run time. Moreover, the nature of the reported information can be adapted dynamically based on changing conditions or directives from another authority, such as an intrusion detection system. The ability to reconfigure the security aspects of components at run time is especially relevant to mobile computing environments. In handheld and wearable systems, constant monitoring of all parameters of interest may be too expensive in terms of computing resources and memory requirements. Rather, certain security checks associated with a component should be loaded only as needed.

We have used Adaptive Java to develop an adaptable audit framework. Besides informers, the framework includes several special types of components. A *decision maker* (DM) controls all of the nonfunctional behavior of the the subcomponents of its container component. Depending on the rules and the current situation within its subsystem, a DM might decide to transform a particular component into an informer by metafying the component with a security-oriented MOP. Once an informer is metafied, its DM can invoke transmutations to start the logging of any part of the internal state of the component. The logged information can be requested later by way of refractions, or the informer can generate an event to report items of interest. Our prototype system includes *information event mediators* (IEMs), which decouple informer components from those components that may be interested in a particular event. A listener simply registers its interests with the IEM, and is notified whenever the respective event has fired. Typically, one of the listeners associated with events fired by informers will be a component of an intrusion detection system (IDS).

To test the operation of our system, we conducted an experiment in a MetaSocket is transformed into an informer that detects anomalies in packet streams. In particular, we investigated the use of informer sockets to monitor the behavior of wireless audio channels at run time. We used the same audio streaming application described in Section 3. However, in this experiment we streamed the data to wireless iPAQ handheld computers, instead of Xybernaut wearables. Moreover, we introduced a second "malicious" source of packets to the receivers. Figure 9 shows the physical configuration of this experiment, where again interactive audio is sent over a wireless LAN from a workstation to multiple iPAQ handheld computers. As before, each packet contains 256 bytes, or 32 msec of audio. This interpacket delay at the sender (and implicitly, the delay between packets arriving at the receivers) stabilizes soon after the channel is established. Hence, any significant changes in this rate indicates either a malfunction or possible malicious behavior.

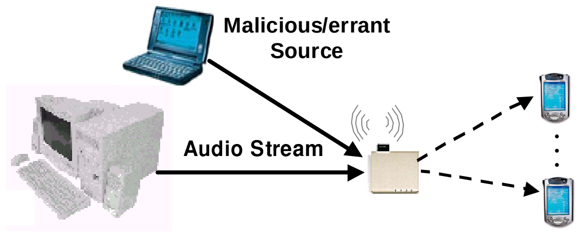


Figure 9: Physical experimental configuration.

When the application begins execution, the DM for the MetaSocket at the receiver first calculates the initial expected rate of arriving packets, based on input parameters to the application. The DM then metafies the MetaSocket as a self-informer, creating several refractions (`isInforms`, `getArrivalTimeVector`, `isNoiseDetected` and `getPacketTimeTolerance`) and transmutations (`setInforms`, `resetArrivalTimeVector`, and `setPacketTimeTolerance`). The DM then uses `setPacketTimeTolerance()` to set the expected packet rate and an error tolerance. From this point, whenever the self-informer MetaSocket detects a significant change in the arrival rate of the packets, the MetaSocket will generate an event (NoiseDetectedEvent).

At a predetermined point in the experiment, we started the second source of packets to the multicast address used by the iPAQs. The informer socket monitors the slope of the packet arrival curve, shown in Figure **??**. The trace of the interpacket delay is shown in Figure 10. When it detects a significant difference (as defined by the application) in the slope for a specified interval of time, it fires the event. In the experiment, additional packets begin arriving after packet number 349. In this experiment, we simply used a second source with the same packet size and interpacket delay as the original source, resulting in a mean interpacket delay of approximately 16 milliseconds. The filter detects these noise packets at packet number 379, which is promising. Of course, we might be able to do even better, but we need to be careful not to fire false events to "normal" variation in the packet arrival time.

The MetaSocket reports the event to the IEM, which forwards it the DM for the MetaSocket. A rule in the DM indicates that an additional filter, NoiseRemover, should be inserted in the MetaSocket. (Another rule could send an alarm message to an administrator, although we did not implement this behavior.) The DM requests the filter from the Component Loader, which forwards the request to the Trader. In the current implementation, a simple unique identifier, rather than a semantic description, is used to reference filters.

The DM inserts the returned filter into the MetaSocket using the transmutation `insertFilter`, mentioned in Section **??**. The NoiseRemover filter uses a simple algorithm to identify packets not associated with the original stream. Specifically, the filter removes any packet whose application-level sequence number is outside a small window. The mean interpacket delay quickly returns to 32 milliseconds. In this example, we assume the data is encrypted and that it would be unlikely for an intruder, or an errant packet source, to send packets with sequence numbers matching the original stream. Those packets identified as noise are dropped, while the others are passed to the receiving application.

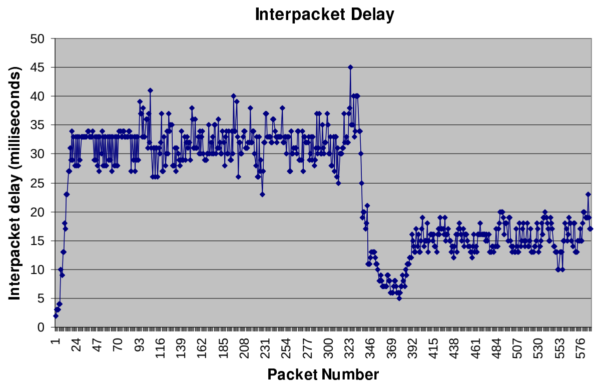


Figure 10: Trace of interpacket delay. A malicious source starts at packet number 349, and the MetaSocket detects the anomaly at packet number 379.

The reader might wonder why a developer would write MetaSocket code that could deliver packets out of order to the application. Moreover, why do we need such a complicated framework simply to filter noise packets from a data channel that exhibits well-defined properties? Actually, these questions highlight precisely the advantages of an adaptive run-time auditing framework. Prior to metafication, the base-level socket component is simply responsible for receiving packets and delivering them to the application. It does not, and should not, care about application-level semantics, including sequence numbers. Moreover, the original developer might not have anticipated all possible audits on that component. In the language of aspect-oriented programming [20], the cross-cutting security code has been detangled from the code that implements the functional behavior of the component. When the MetaSocket is created through metafication, some basic checks are inserted into the component, specific to its use for audio streaming. However, to further inspect the data in the packet, including the application-level header, is not necessarily warranted and would likely waste computing resources. The proposed techniques enable all auditing functionality to be completely tailorable to current conditions and added only as needed.

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 [11], MOST [14], Rover [17], MASH [27], TAO [23], dynamicTAO [21], MobiWare [4], MCF [24], QuO [43], MPA [37], Odyssey [34], DaCapo++ [40], RCSM [45], 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 [22], Puppeteer [12], Aura [39], 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 [20]. Although many projects in the AOP community focus on compile-time weaving [19], 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 [7], 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

A major goal of the RAPIDware project is to explore software mechanisms that enable coordinated adaptation to changing conditions in multiple cross-cutting concerns: security, energy consumption, fault tolerance, and quality of service. In this paper, we described the use of the Adaptive Java programming language to support the development of components that can be adapted in multiple ways. Specifically, we showed how MetaSocket components can be adapted at run time for error control, energy management, and security auditing. While the experiments conducted are relatively simple, they serve as a proof-of-concept that it is possible to transform components at run time such that they are adaptable in different dimensions. Moreover, while we focused exclusively on MetaSockets in this paper, we emphasize that the Adaptive Java mechanisms are general. Currently, we are conducting several additional RAPIDware 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; and mitigation of the heterogeneity of system display characteristics. In addition, we are developing additional language constructs that facilitate capturing the relevant state of a component in order to support dynamic changes in fault-tolerance requirements. Combined, these activities are intended to provide developers with tools and techniques to develop adaptive software that can take advantage of compile-time and run-time correctness checking.

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`.

Acknowledgements. The authors would like to thank Kurt Stirewalt, Rahul Kalaskar, and Udiyan Padmanabhan for their contributions to this work. 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.

8. 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] J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff, E. Spafford, and D. Zamboni. Aspects for run-time component integration. In *Proceedings of the 14th Annual Computer Security Applications Conference*, pages 13–24, December 1998.

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

[8] 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.

[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] 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.

[12] 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.

[13] A. Fox, S. D. Gribble, Y. Chawathe, and E. A. Brewer. Adapting to network and client variation using active proxies: Lessons and perspectives. *IEEE Personal Communications*, August 1998.

[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] W. Jansen, P. Mell, T. Karygiannis, and D. Marks. Mobile agents in intrusion detection and response. In *Proceedings of the 12th Annual Canadian Information Technology Security Symposium*, Ottawa, Canada, 2000.

[17] 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.

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

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

[26] A. J. McAuley. Reliable broadband communications using burst erasure correcting code. In *Proceedings of ACM SIGCOMM*, pages 287–306, September 1990.

[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] T. Qian and R. Campbell. Dynamic agent-based security architecture for mobile computers. In *Proceedings of the International Conference on Parallel and Distributed Computing and Networks (PDCN'98)*, December 1998.

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

[37] 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.

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

[39] 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.

[40] 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.

[41] A. Tripathi, T. Ahmed, S. Pathak, A. Pathak, M. Carney, M. Koka, and P. Dokas. Active monitoring of network systems using mobile agents. In *Proceedings of Networks 2002, Joint Conference of ICWLHN 2002 and ICN 2002*, August 2002. to appear.

[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] J. Viega and D. Evans. Separation of concerns for security. In *Proceedings of the ICSE Workshop on Multidimensional Separation of Concerns in Software Engineering*, June 2000.

[45] 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.