Generation of Self-Optimizing Wireless Network Applications (Poster Summary)

S. M. Sadjadi, P. K. McKinley, R. .E. K. Stirewalt, and B. H. C. Cheng
Software Engineering and Network Systems Laboratory
Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan 48824
{sadjadis,mckinley,stire,chengb}@cse.msu.edu

Motivations. As the computing and communication infrastructure continues to expand and diversify, managing the software developed for such heterogeneous environments is becoming more and more complex; therefore, the need for autonomic computing [1] is increasing. As one example, the advent of the "Mobile Internet" has produced a variety of handheld and wearable computing devices, whose software must adapt to multiple, potentially conflicting concerns, such as quality-of-service, security, and energy consumption. Unfortunately, many applications being ported to mobile computing environments were not designed to be adaptable to the corresponding concerns. We say that a program is *adaptable* if it contains facilities for selecting and incorporating new behaviors at run time.

Adaptable applications are difficult to develop and maintain especially if *adaptive* code, which implements the adaptive behavior, is tangled into *functional* code, which implements the imperative behavior. Adaptive code that deals with concerns such as quality of service tends to crosscut the conventional "functional decomposition" of an application used for the functional code. Therefore, modifying existing programs directly to support adaptation can lead to tangled code that is error-prone and difficult to maintain. In the past few years, numerous techniques have been proposed that enable separation of adaptive code from the functional code [2]. Kephart and Chess [1] describe an architecture for autonomic elements that promotes this separation of concerns. They suggest a structure for autonomic elements that consists of one or more *manageable elements*, which implement the imperative behavior, coupled with a single *autonomic manager*, which implements the adaptive behavior and controls the managed elements.

Our Approach. Our work investigates how to wrap existing applications transparently to generate such manageable components. Developing manageable components requires some mechanism to support adaptation in behavior. The predominant mechanism for implementing adaptation in object-oriented software is *behavioral reflection*, which can be used to modify how an object responds to a message. In recent years, behavioral reflection has been used to support adaptation to a variety of different concerns, including quality-of-service and fault tolerance. Unfortunately, programs that use behavioral reflection incur additional overhead, as in some cases every message sent to an object must be intercepted and possibly redirected. To provide efficient autonomy, a developer should be able to *selectively* introduce behavioral reflection only where needed to support the desired adaptations. Specifically, the developer should be able to identify, at compile time, which individual classes should be able to support adaptation at run time.

We previously showed how to use aspect-oriented programming to selectively introduce behavioral reflection into an existing program [3]. However, the reflection used there is *ad hoc* in that the developer must invent the reflective support classes and supporting infrastructure for adaptation, and must create an aspect that weaves this infrastructure into the existing program. More recently, we developed transparent reflective aspect programming (TRAP) [4], which combines behavioral reflection [5] and aspect-oriented programming [6] to transform extant programs into manageable programs without the need to modify the original source code manually. TRAP supports general behavioral reflection by automatically generating wrapper classes and meta-classes from selected classes in an application. TRAP then generates aspects that replace instantiations of selected classes with instantiations of their corresponding wrapper classes. This two-pronged, automated approach enables powerful behavioral reflection with minimal overhead. To validate these ideas, we developed TRAP/J, which instantiates TRAP for Java programs [4].

Case Study. This work describes the application of TRAP/J to a multicast audio application. The Audio-Streaming Application (ASA) is designed to deliver in-

Proceedings of the International Conference on Autonomic Computing (ICAC'04)
0-7695-2114-2/04 $20.00 © 2004 IEEE

teractive audio from a microphone at one network node to multiple receiving nodes. Although the original application was developed for wired networks, we used TRAP/J to make it adaptable to wireless environments, where the packet loss rate is dynamic and location dependent. In our experiments, a laptop workstation multicasts the audio stream to multiple wireless iPAQs over an 802.11b (11Mbps) ad-hoc wireless local area network (WLAN) (Figure 1). In current WLANs, packet losses affect multicast communication more than unicast communication, since the 802.11b MAC layer does not provide link-level acknowledgements for multicast frames.

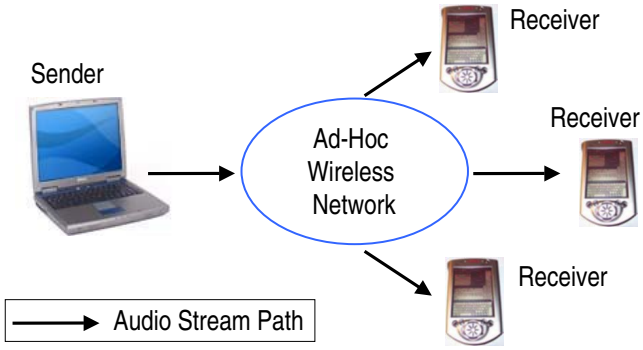

Figure 1: Audio streaming in a wireless LAN.

Figure 2 illustrates the strategy we used to enable ASA to adapt to variable channel conditions in wireless networks. We use TRAP/J to replace normal Java socket functionality with *MetaSocket* [7] functionality, transparently to the ASA code. *MetaSockets* are adaptable communication components created from existing Java socket classes, but their structure and behavior can be adapted at run time in response to external stimuli (*e.g.,* dynamic wireless channel conditions).

The particular MetaSocket adaptation used here is the dynamic insertion and removal of *forward-error correction* (FEC) filters [8]. Specifically, an FEC encoder filter can be inserted and removed dynamically at the sending MetaSocket, in synchronization with an FEC decoder being inserted and removed at each receiving MetaSocket. Use of FEC under high packet loss conditions reduces the packet loss rate as observed by the application. Under low packet loss conditions, however, FEC should be removed so as not to waste bandwidth on redundant data. Experiments on a mobile computing testbed demonstrate the utility of TRAP/J to transparently and automatically enhance an existing application with new adaptive behavior, specifically, enabling ASA to better deal with highly variable conditions in wireless networks.

Further Information. A number of related papers and technical reports of the Software Engineering and Network

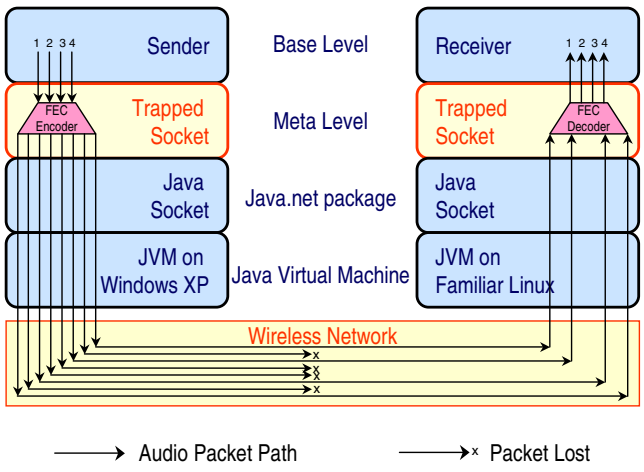

Figure 2: Adaptation strategy.

Systems Laboratory can be found at the following URL: `http://www.cse.msu.edu/sens`.

Acknowledgements. 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 CCR-9912407, EIA-0000433, EIA-0130724, and ITR-0313142.

References

[1] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[2] K. Czarnecki and U. Eisenecker, *Generative programming*. Addison Wesley, 2000.

[3] Z. Yang, B. Cheng, R. Stirewalt, J. Sowell, S. M. Sadjadi, and P. K. McKinley, "An aspect-oriented approach to dynamic adaptation," in *Proceedings of the ACM SIGSOFT Workshop On Self-healing Software*, Nov. 2002.

[4] S. M. Sadjadi, P. McKinley, R. Stirewalt, and B. Cheng, "TRAP: Transparent reflective aspect programming," Tech. Rep. MSU-CSE-03-31, Department of Computer Science, Michigan State University, East Lansing, Michigan, November 2003.

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

[6] 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.

[7] S. M. Sadjadi, P. K. McKinley, and E. P. Kasten, "Architecture and operation of an adaptable communication substrate," in *Proceedings of the Ninth International Workshop on Future Trends of Distributed Computing Systems (FTDCS '03)*, May 2003.

[8] L. Rizzo and L. Vicisano, "RMDP: An FEC-based reliable multicast protocol for wireless environments," *ACM Mobile Computer and Communication Review*, vol. 2, April 1998.