Transparent Self-Optimization in Existing CORBA Applications *

S. M. Sadjadi and P. K. McKinley
Software Engineering and Network Systems Laboratory
Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan 48824
{sadjadis,mckinley}@cse.msu.edu

Abstract

This paper addresses the design of adaptive middleware to support autonomic computing in pervasive computing environments. The particular problem we address here is how to support self-optimization to changing network connection capabilities as a mobile user interacts with heterogeneous elements in a wireless network infrastructure. The goal is to enable self-optimization to such changes transparently *with respect to the core application code. We propose a solution based on the use of the* generic proxy*, which is a specific CORBA object that can intercept and process any CORBA request using rules and actions that can be introduced to the knowledge base of the proxy during execution. To explore its design and operation, we have incorporated the generic proxy into ACT [1], an adaptive middleware framework we designed previously to support adaptation in CORBA applications. Details of the generic proxy are presented. A case study is described in which a generic proxy is used to support self-optimization in an existing image retrieval application, when executed in a heterogeneous wireless environment.*

Keywords: *adaptive middleware, autonomic computing, self-optimization, dynamic adaptation, transparent adaptation, generic proxy, quality-of-service, mobile computing, CORBA.*

1. Introduction

Applications executing in pervasive computing environments need to adapt to dynamic conditions in a variety of ways. The need for adaptation arises in part due to (1) the variable nature of wireless network channels (2) the heterogeneity of the wireless infrastructure, and (3) the heterogeneity of computing platforms. Let us consider just the simple case of a single mobile device as it interacts with the wireless infrastructure. The infrastructure may comprise many different wireless cells, possibly managed by different organizations, with different physical characteristics and different management/security policies. Moreover, the cells may not always overlap, resulting in temporary (or longer) periods of disconnection. As the user moves from cell to cell, an autonomic system should attempt to minimize the disruption to the applications executing on the mobile device and should provide smooth transitions, with minimal user intervention, when disruptions are unavoidable.

Solving this problem is complicated by the fact that the nature of self-management is often domain- or application-specific. While new applications can be designed to accommodate these dynamics directly by being context-aware [2], self-management is also needed in legacy applications that are executed in new environments. Moreover, even if some aspects of the dynamic environment were anticipated when the application was developed, others are likely to arise after it is deployed. Hence, it may be desirable to *separate* code that implements adaptive behavior from code that implements the imperative behavior of the application [3]. Kephart and Chess [4] 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 work investigates how such separation of concerns can enable applications to self-adapt to their environments in ways that were not necessarily anticipated at development time, and in ways that are *transparent* to the imperative core of the application code. To accomplish this task, we employ an *adaptive middleware* approach. Just as tra-

* 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.

ditional middleware hides distribution from an application, so can adaptive middleware hide automatic adaptation to changes in the execution environment, such as network connectivity, energy usage, and security policies [5–13]. Some adaptations are "generic," in that they are common to many, if not all, applications executing on the device. For example, a new IP address may be assigned to a device that has been temporarily disconnected when it reconnects in a new wireless domain, and such a change should be transparent to most applications running on that device. Some adaptations are specific to the application. For example, a conferencing application may adapt the quality of audio/video streams, or may eliminate some streams entirely, in order to accommodate a sudden drop in available bandwidth. To enable both types of adaptations in a transparent manner, it must be possible to introduce both application-specific and generic functionality to the middleware as it executes.

In this paper, we propose and evaluate a possible solution to this problem, the *generic proxy*. The generic proxy is a particular CORBA object that can receive any CORBA request. The generic proxy can be introduced to a middleware platform at run-time to enable intercepted communication streams to be processed in either a generic or application-specific way. We have integrated the generic proxy into ACT [1], an adaptive middleware framework we introduced previously to support unanticipated adaptation in CORBA applications. An ACT-based framework can be coupled with a CORBA application transparently at run time: new types of adaptation can be added without recompiling the application. In [1], we described the basic architecture and operation of ACT (reviewed in Section 2) and showed how it can be used to support unanticipated adaptation. However, the original ACT architecture required a separate proxy for each type of intercepted request. In this paper, we report how the generic proxy can used in place of any specific proxy. This capability is particularly useful in pervasive computing contexts, where many situations (e.g., different security policies and disparate capabilities among wireless domains) require adaptation irrespective of the application.

The main contributions of this paper are threefold. First, we present the design and implementation of the generic proxy, focusing on the key issue in its operation, namely, its interaction with the CORBA interface repository [14] in order to support interception of multiple types of requests. Second, we propose and evaluate a design for a flexible rule-based decision maker, which is used by the generic proxy to determine the fate of a particular CORBA request. Third, we demonstrate the role that the generic proxy can play in pervasive computing by enabling transparent, unanticipated self-management. Specifically, we describe a case study in which we used the generic proxy in ACT to implement transparent self-optimization in an existing CORBA application, enabling it to accommodate changing conditions and interacts with heterogeneous components of a wireless network infrastructure.

The remainder of the paper is organized as follows. Section 2 reviews the architecture and operation of ACT. Section 3 describes the generic proxy and how it is incorporated within the ACT framework. Section 4 describes the case study. Section 5 discusses related work, and Section 6 summarizes the paper and mentions possible future directions.

2. ACT Background

Schmidt [15] decomposes middleware into four layers: host-infrastructure, distribution, common-services, and domain-services. Figure 1 illustrates these layers. Since the operation of ACT involves all four, we provide a brief overview here.

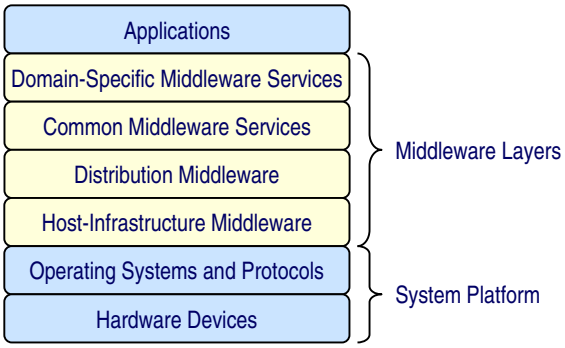
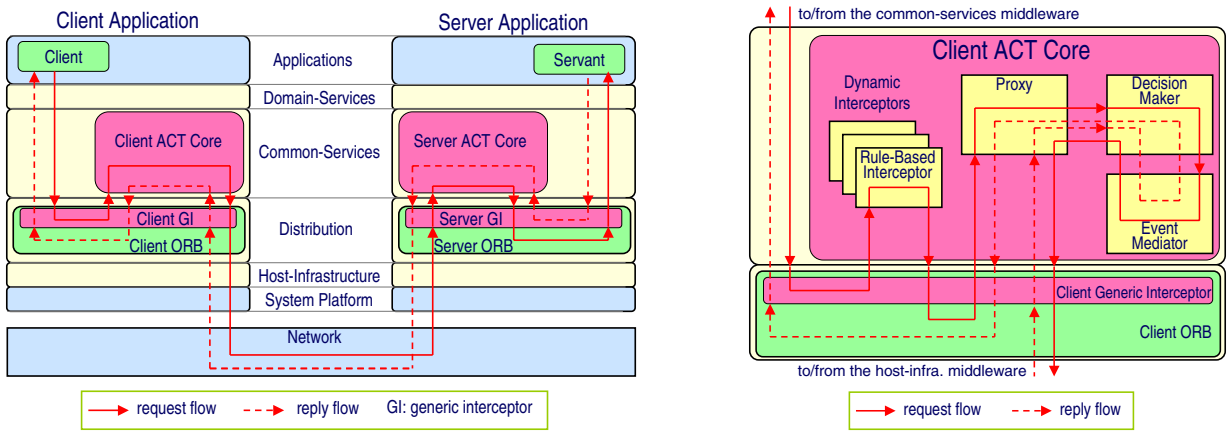


Figure 1: Middleware layers defined by Schmidt [15].

The host-infrastructure layer resides directly atop the operating system and provides a higher-level API that hides the heterogeneity of hardware platforms, operating systems and, to some extent, network protocols. The host-infrastructure layer provides generic services to the upper middleware layers by encapsulating functionality that would otherwise require many tedious, error-prone, and non-portable code, such as socket programming and thread communication primitives. ACE [16] and Rocks [17] are examples of adaptive middleware in this layer. *The distribution layer* resides atop the host-infrastructure layer and provides high-level programming abstractions, such as remote object operations, to the developer. Using the distribution layer, a developer can write a distributed application in a similar way to a stand-alone application. In addition, this layer hides the heterogeneity of network protocols and, in some cases, the heterogeneity of operating systems and programming languages. CORBA [14], DCOM [18], and Java RMI [19] are the main solutions to distribution middleware. Adaptive ORBs, which reside in this layer, include TAO [12], DynamicTAO [5], ZEN [13], OpenORB [6], and



(a) ACT component configuration

(b) ACT Core components

Figure 2: ACT architecture [1].

Electra [20]. *The common-services layer* resides atop the distribution layer and provides services such as fault tolerance, security, load balancing, event propagation, logging, persistence, real-time scheduling, and transactions. Examples of adaptive CORBA frameworks that provide such services include QuO [8], IRL [21], and FRIENDS [22]. *The domain-specific layer* resides atop the common-services layer and is tailored to a specific class of distributed applications. Unlike the common-services layer, the services in this layer can be reused only for a specific domain of applications. The Boeing Bold Stroke architecture [23] is an example of adaptive middleware in this layer that benefits from the capabilities of real-time CORBA ORBs and supports configurable and reusable avionics services.

By coordinating actions at various middleware layers, ACT (short for Adaptive CORBA Template) supports transparent enhancements, either at startup time or during run-time, to CORBA applications. Specifically, ACT enables CORBA applications to adapt to previously unanticipated changes in their functional requirements or in non-functional concerns, such as quality-of-service, fault-tolerance, and security. The key insight into how to achieve this transparency is the concept of the *generic interceptor*, which is a particular type of CORBA portable request interceptor [14]. The generic interceptor provides a "hook" into the interaction between CORBA clients and servants, enabling the insertion of new adaptive functionality after deployment. Although the generic interceptor must itself be registered with the ORB of a CORBA application at startup time, it enables registration of other *specific* request interceptors to be postponed until run time. The interceptors can adapt requests, replies, and exceptions that pass through the ORB. As a result, an existing application need only be restarted with an argument identifying a generic proxy object to be used. There is no need to modify or even recompile the application. At run time, the generic proxy can be used to incorporate specific interceptors that dynamically adapt the application behavior.

Figure 2(a) shows the flow of a request/reply sequence in a simple CORBA application using ACT. For clarity, details such as CORBA stubs and skeletons are not shown, although we do identify the middleware layer(s) [15] in which each component resides. The *client* generic interceptor intercepts all outgoing requests and incoming replies (or exceptions) and forwards them to its *ACT core*. Similarly, the generic interceptor at the server side intercepts all the incoming requests and outgoing replies (or exceptions) and forwards them to its ACT core.

Figure 2(b) shows the flow of a request/reply sequence within the client ACT core. The request is first processed by one or more *dynamic* interceptors. Unlike the generic interceptor, which is statically configured at start-up time, dynamic interceptors can be registered after the ORB initialization time (at run time) with the generic interceptor. A *rule-based* interceptor (RBI) is an example of dynamic interceptors that uses "rules" to govern its operation. The rules can be inserted, removed, and modified at run time. A *proxy* is a surrogate for a CORBA object that provides the same set of methods as the CORBA object. A proxy consults with a *decision maker* in determining how to handle intercepted requests. Possibilities include sending a new request (possibly with modified arguments) to either the target object or to another object. Alternatively, and unlike a request interceptor, a proxy can reply to the intercepted requests using local data (*e.g.,* cached replies). In the case of exceptional situa-

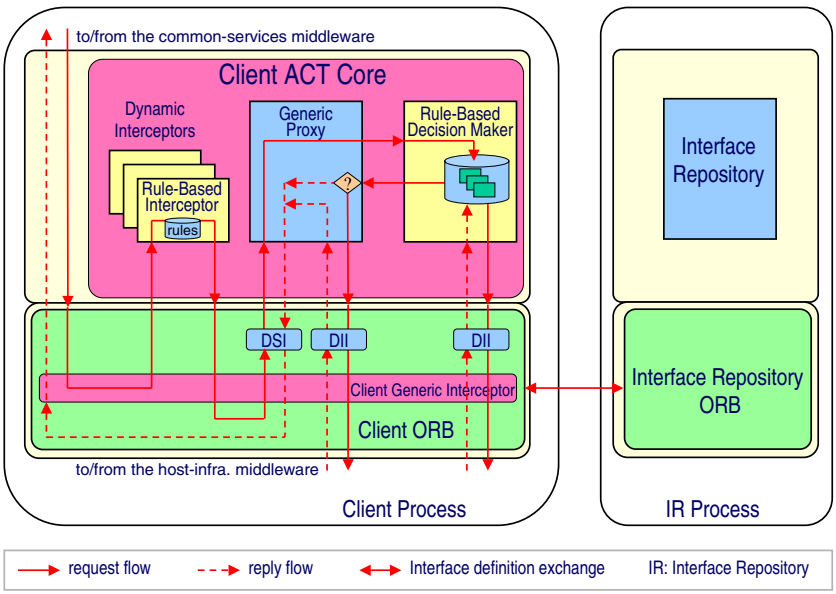


Figure 3: Incorporating generic proxy in the ACT core.

tions, the decision maker may notify other objects by way of an *event mediator* [24], which propagates events to interested objects.

We refer to ACT as a framework *template*, because it provides a generic model for constructing and enhancing adaptive CORBA frameworks. Several such frameworks have been developed recently to support quality-of-service [8], real-time processing [12, 13], fault tolerance [21, 25], and mobile computing [26–28]. ACT can be used to develop an object-oriented framework in any programming language that supports dynamic loading of code and can be applied to any CORBA ORB that supports portable interceptors [14]. We previously developed a Java prototype of ACT, as well as a set of administrative consoles that enable manual adaptation of applications at run time. The prototype uses ORBacus, a CORBA-compliant ORB distributed by IONA Technologies. To demonstrate the seamless interaction of ACT with other adaptive CORBA frameworks, we combined ACT with the QuO framework [8] developed at BBN Technologies. The resulting framework is able to weave quality-of-service (QoS) aspects, referred to as *qoskets* in QuO terminology, into CORBA applications both at compile time and at run time.

In the remainder of this paper, we describe how ACT can be used to construct a Java-based adaptive framework to support autonomic computing in pervasive computing environments. We begin with a discussion of how the generic proxy helps achieve this goal, followed by a case study in which we demonstrate transparent self-optimization in an existing image retrieval application.

3. Generic Proxy

To enable dynamic weaving of adaptive functionality that is common to multiple applications, ACT needs to intercept and adapt CORBA requests, replies, and exceptions in a manner independent of the semantics (the application logic) and syntax (the CORBA interfaces defined in the application) of specific applications. The *generic proxy* is a particular CORBA object that is able to receive *any* CORBA request (hence the label "generic"). To determine how to handle a particular request, the generic proxy accesses the CORBA interface repository [14], which provides all the IDL descriptions for CORBA requests. The repository executes as a separate process and is usually accessed through the ORB. Most CORBA ORBs provide a configuration file or support a command-line argument that allows the user to introduce the interface repository to the application ORB. Providing IDL information to the generic proxy in this manner implies no need to modify or recompile the application source code. The interface repository, however, requires access to the CORBA IDL files used in the application.

In default operation, the generic proxy intercepts CORBA requests, acquires the request specifications from a CORBA interface repository, creates similar CORBA requests and sends them to the original targets, and forwards replies from those targets back to the original clients. A generic proxy also publishes a CORBA service that can be used to register a *decision maker*.

Figure 3 illustrates the sequence of a request/reply in the ACT core that contains a rule-based interceptor, a generic proxy, and a rule-based decision maker. First, a request from

the client application is intercepted by the rule-based interceptor, which checks its rules for possible matches. A default rule, initially inserted in its knowledge base, directs the rule-based interceptor to raise a ForwardRequest exception, which results in its forwarding the request to the generic proxy. When the generic proxy receives the request, it acquires the request interface definition via the application ORB, which in turn retrieves the information from the interface repository. The proxy creates a new request and forwards it to the rule-based decision maker. The rule-based decision maker checks its knowledge base for possible matches to the request. Depending on the implementation of the rules, the decision maker may return either a modified request to the generic proxy or a reply to the request. If the decision maker returns the request (or a modified request), the generic proxy will continue its operation by invoking the request. If the reply to the request is returned by the decision maker, the proxy replies to the original request using the reply from the decision maker. The generic proxy uses the CORBA dynamic skeleton interface (DSI) [14] to receive any type of request. The generic proxy and the rule-based decision maker use the CORBA dynamic invocation interface (DII) [14] to create and invoke a new request dynamically.

4. Case Study: Adaptive Mobility Management

To evaluate the effectiveness of ACT to support self-management in existing CORBA applications, without modifying the application code, we conducted a case study in which self-optimization is enabled in an existing application. Additional experiments involving IP handoff, are described in an accompanying technical report [29]. We begin with a brief overview of the application and the experimental environment, followed by the description of the experiment. The experiment shows how ACT could be used to support autonomic computing in either a generic or application-specific manner.

4.1. The Example Application and Experimental Environment

For the application, we adopted a distributed image retrieval application developed by BBN Technologies, which we also used in our first ACT study [1]. The application has two parts, a client that requests and displays images, and a server that stores the images and replies to requests for them. In this study, we treat the application as though it is used for surveillance, with a mobile user executing the client code on a laptop and monitoring a physical facility through continuous still images from multiple camera sources. For the experiment described later in this section, we executed the server on a desktop computer connected to a 100 Mbps wired network and the client on a laptop computer connected to a three-cell 802.11b wireless network. Both the desktop and laptop systems are running the Linux operating system.

Figure 4 shows the physical configuration of the three access points used in the experiment. (The wireless cells are drawn as circles for simplicity – the actual cell shapes are irregular, due to the physical construction of the building and orientation of antennas.) AP-1 and AP-3 provide 11Mbps connections, whereas AP-2 provides only 2Mbps. The desktop running the server application is close to AP-1. AP-1 and AP-2 are managed by our Computer Science and Engineering Department, whereas AP-3 is managed by the College of Engineering. This difference implies that the IP address assigned to the client laptop needs to change as the user moves from a CSE wireless cell to a College cell.

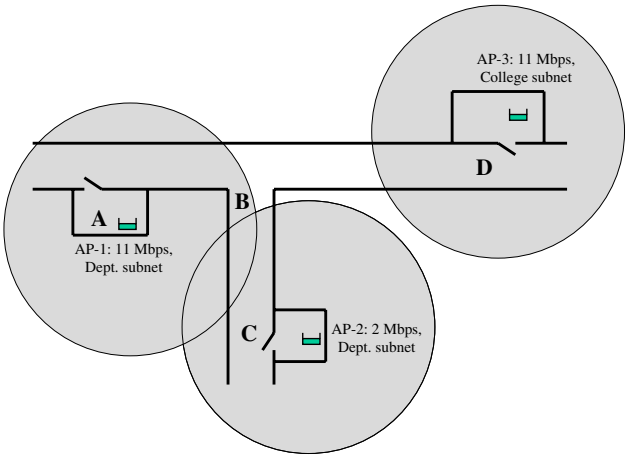


Figure 4: The configuration of the access points used in the experiment.

Figure 5 shows an example image from the experiment. The server provides four different versions of each image, varying in size and quality. Typical comparative file sizes are 90KB, 25KB, 14KB, and 4KB.

4.2. Self-Management and Self-Optimization

To investigate how ACT can support self-management, we developed an application-specific rule that maintains the frame rate of the application by controlling the image size or inserting inter-frame delays dynamically. The original image retrieval application operates in a default mode, which retrieves and plays images as fast as possible. ACT enables us to weave the rule into the application at run-time, thereby providing new functionality (frame rate control) transparently with respect to the application. The self-optimization rule maintains the frame rate of the application in the pres-

Figure 5: An image from the experiment.

ence of dynamic changes to the wireless network loss rate, the network (wired/wireless) traffic, and CPU availability.

Figure 6 shows the Automatic Adaptation Console, which diplays the application status and also enables the user to user to enter quality-of-service preferences. As shown in this figure, the rule uses several parameters to decide on when and how to adapt the application in order to maintain the frame rate. These parameters have default values as shown in the figure, but can be modified at run time by the user. The Average Frame Rate Period indicates the period during which the average frame rate should be calculated to be considered for adaptation. The Stabilizing Period specifies the amount of time that the rule should wait until the last adaptation stabilizes; also if a sudden change occurs in the environment such as hand-off from one wireless cell to another one, the system should wait for this period before it decides on the stability of the system. The rule detects a stable situation using the Acceptable Rate Deviation; when the frame rate deviation goes below this value, the system is considered stable. Similarly, the rule detects an unstable situation, if the instantaneous frame rate deviation goes beyond the Unacceptable Rate Deviation value. The rule also maintains a history of the round-trip delay associated with each request in each wireless cell. Using this history and the above parameters, the rule can decide to maintain the frame rate either by increasing/decreasing the inter-frame delay or by changing the request to ask for a different version of the image with smaller/larger size. The default behavior of the rule is to display images that are as large as possible, given the constraints of the environment.

Figure 7 shows a trace demonstrating automatic adaptation of the application in the following scenario. In this experiment, the user has selected a desired frame rate of 2 frames per second, as shown in Figure 6. For the first 60 sec-

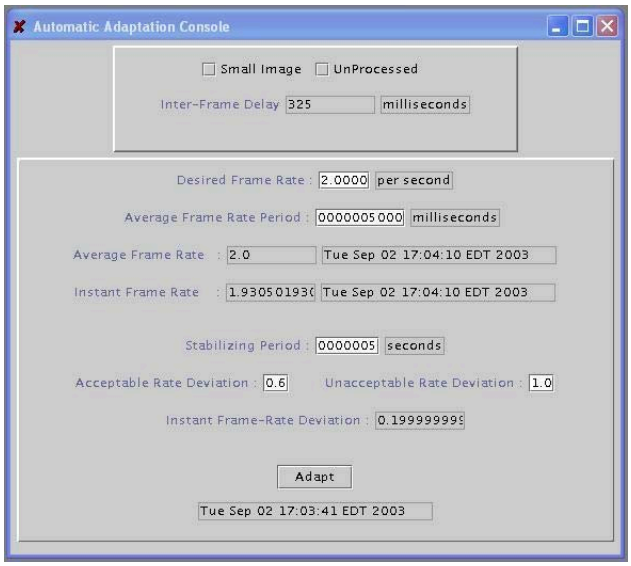


Figure 6: Automatic Adaptation Console.

onds of the experiment, the user stays close to the location A (Figure 4). The rule detects that the desired frame rate is lower than the maximum possible frame rate, based on observed round-trip times. Hence, it inserts an inter-frame delay of approximately 200 milliseconds to maintain the frame rate at about 2 frames per second. At point 120 seconds, the user starts walking from location A to location B for 60 seconds. The automatic adaptation rule maintains the frame rate by decreasing the inter-frame delay during this period. At point 180 seconds, the user begins walking from location B to location C and back again, returning to location B at 360 seconds. During this period, because the AP-2 access point provides 2Mbps, the automatic adaptation rule detects that the current frame rate is lower than that desired. It first removes the inter-frame delay, but the frame rate does not reach to 2 frames per second. Therefore, it reduces the quality of the image by asking for a smaller image size. Now the frame increases beyond that desired, so the automatic adaptation rule inserts an inter-frame delay of 400 milliseconds to maintain the frame rate at 2 frames per second. Although there is some oscillation, the rate stabilizes by time 360 seconds. At this point, the user continues walking from location B to location A, prompting the rule to reverse the actions. First the inter-frame delay is increased to maintain the frame rate, followed by an increase in image size. In this manner, the rule brings the application back to its original behavior. Again, because the current frame rate is higher that expected, an inter-frame delay of about 200 milliseconds is inserted to maintain the frame rate at 2 frames per second.

This result is promising and demonstrates that it is possible to add self-management behavior to an application transparently to the application code. Moreover, the use

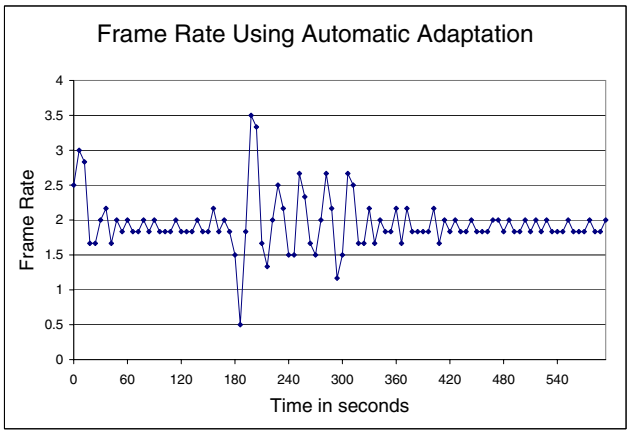


Figure 7: Maintaining the application frame rate using automatic adaptation.

of a generic proxy enables self-optimization functionality, both application-independent and application-specific, to be added to the application, even at run time.

5. Related Work

ACT is intended to complement adaptive middleware frameworks and to support interoperation among incompatible frameworks. Like ACT, many adaptive middleware projects use the aspect-oriented programming (AOP) paradigm [3], facilitating separation of functional aspects from non-functional aspects (*e.g.,* self-management, quality-of-service, security, and fault-tolerance). Examples include QuO [8], AspectIX [9], and Squirrel [10]. Other projects [5, 6] use computational reflection [30] as the primary mechanism for adaptation, enabling middleware to inspect and modify its structure and behavior during execution. Examples include DynamicTAO [5], OpenORB [6], Iguana/J [31], and ZEN [13]. By transparently intercepting requests, replies, and exceptions, ACT could serve as a *framework gateway*, enabling applications developed under such frameworks to interact seamlessly.

The concept of transparently intercepting CORBA requests and replies has been used in several projects. Friedman et al. [32] use CORBA portable interceptors to enhance the client side of a CORBA application by introducing proxies that can cache replies and forward requests to other CORBA objects. This work is among the first to exploit CORBA portable interceptors for transparent adaptation. In the IRL project, Baldoni et al. [21] use portable interceptors to transparently introduce their implementation of fault-tolerant CORBA [14] to CORBA-compliant ORBs. Moser et al. [25] also use an interception-based approach in Eternal, which transparently introduces afault-tolerant CORBA implementation to CORBA applications. Eternal, however, employs an operating-system interception-based approach instead of CORBA portable interceptors. In general, the above projects focus on modifying program behavior in a particular way, for example, to enhance fault tolerance. Like ACT, the DADO project [33] uses CORBA portable interceptors to support dynamic weaving of multiple cross-cutting concerns such as security, fault tolerance, and QoS. However, ACT uses the concept of *generic* interceptors to enable late binding of the adaptation infrastructure itself. Moreover, generic interception enables ACT to be used as a framework gateway.

6. Conclusions

In this paper, we introduced the generic proxy, which supports autonomic computing in existing CORBA applications transparently to the application code. We described details of the generic proxy in the context of ACT, an interception-based adaptive middleware frameworks. Finally, in the context of a case study, we showed how a generic proxy can be incorporated in ACT to support self-optimization related to the dynamic behavior of wireless infrastructure in pervasive computing environments. In this manner, the generic proxy enables mechanisms that facilitate pervasive computing by supporting autonomy in existing CORBA applications, without the need to modify or recompile their source code, and in ways not anticipated during development.

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] S. M. Sadjadi and P. K. McKinley, "ACT: An adaptive CORBA template to support unanticipated adaptation," in *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS'04)*, (Tokyo, Japan), March 2004.

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

[3] 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.

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

[5] F. Kon, M. Román, P. Liu, J. Mao, T. Yamane, L. C. Magalhães, 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.

[6] 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.

[7] M. Astley, D. C. Sturman, and G. A. Agha, "Customizable middleware for modular distributed software," *Communications of the ACM*, vol. 44, no. 5, pp. 99–107, 2001.

[8] J. A. Zinky, D. E. Bakken, and R. E. Schantz, "Architectural support for quality of service for CORBA objects," *Theory and Practice of Object Systems*, vol. 3, no. 1, 1997.

[9] M. Geier, M. Steckermeier, U. Becker, F. J. Hauck, E. Meier, and U. Rastofer, "Support for mobility and replication in the AspectIX architecture," Tech. Rep. TR-I4-98-05, Univ. of Erlangen-Nuernberg, IMMD IV, 1998.

[10] R. Koster, A. P. Black, J. Huang, J. Walpole, and C. Pu, "Thread transparency in information flow middleware," in *Proceedings of the International Conference on Distributed Systems Platforms and Open Distributed Processing*, Springer Verlag, Nov. 2001.

[11] N. Venkatasubramanian, M. Deshpande, S. Mohapatra, S. Gutierrez-Nolasco, and J. Wickramasuriya, "Design and implementation of a composable reflective middleware," in *Proceedings of the 21th International Conference on Distributed Computing Systems (ICDCS-21)*, April 2001.

[12] D. C. Schmidt, D. L. Levine, and S. Mungee, "The design of the TAO real-time object request broker," *Computer Communications*, vol. 21, pp. 294–324, April 1998.

[13] R. Klefstad, D. C. Schmidt, and C. O'Ryan, "Towards highly configurable real-time object request brokers," in *Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, April - May 2002.

[14] Object Management Group, Framingham, Massachusett, *The Common Object Request Broker: Architecture and Specification Version 3.0*, July 2003. Available at http://doc.ece.uci.edu/CORBA/formal/02-06-33.pdf.

[15] D. C. Schmidt, "Middleware for real-time and embedded systems," *Communications of the ACM*, vol. 45, June 2002.

[16] D. C. Schmidt, "The ADAPTIVE Communication Environment: An object-oriented network programming toolkit for developing communication software," *Concurrency: Practice and Experience*, vol. 5, no. 4, pp. 269–286, 1993.

[17] V. C. Zandy and B. P. Miller, "Reliable network connections," in *Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking*, pp. 95–106, September 2002.

[18] Microsoft Corporation, *Microsoft COM Technologies - DCOM*, 2000.

[19] Java Soft, *Java Remote Method Invocation Specification, revision 1.5, JDK 1.2*, Oct. 1998.

[20] S. Maffeis, "Adding group communication and fault-tolerance to CORBA," in *Proceedings of the Conference on Object-Oriented Technologies*, pp. 135–146, 1995.

[21] R. Baldoni, C. Marchetti, A. Termini, "Active software replication through a three-tier approach," in *Proceedings of the 22th IEEE International Symposium on Reliable Distributed Systems (SRDS02)*, (Osaka, Japan), pp. 109–118, October 2002.

[22] J. C. Fabre and T. Perennou, "A metaobject architecture for fault-tolerant distributed systems: The FRIENDS approach," *IEEE Transactions on Computers*, vol. 47, no. 1, pp. 78–95, 1998.

[23] D. Sharp, "Reducing avionics software cost through component-based product line development," in *Proceedings of the Software Technology Conference*, (Salt Lake City, Utah), April 1998.

[24] S. M. Sadjadi, P. K. McKinley, and E. P. Kasten, "Architecture and operation of an adaptable communication substrate," in *Proceedings of the Ninth IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'03)*, (San Juan, Puerto Rico), pp. 46–55, May 2003.

[25] L. Moser, P. Melliar-Smith, P. Narasimhan, L. Tewksbury, and V. Kalogeraki, "The Eternal system: An architecture for enterprise applications," in *Proceedings of the Third International Enterprise Distributed Object Computing Conference (EDOC'99)*, July 1999.

[26] M. Haahr, R. Cunningham, and V. Cahill, "Supporting CORBA applications in a mobile environment," in *Proceedings of the Fifth ACM/IEEE International Conference on Mobile Computing and Networking*, 1999.

[27] S. Adwankar, "Mobile CORBA," in *Proceedings of the International Symposium on Distributed Object and Applications (DOA 2001)*, 2001.

[28] M. Liljeberg, K. Raatikainen, M. Evans, S. Furnell, N. Maumon, E. Veldkamp, B. Wind, and S. Trigila, "Using CORBA to support terminal mobility," in *Proceedings of the TINA Conference*, November 1997.

[29] S. M. Sadjadi and P. K. McKinley, "Supporting transparent and generic adaptation in pervasive computing environments," Tech. Rep. MSU-CSE-03-32, Department of Computer Science, Michigan State University, East Lansing, Michigan, November 2003.

[30] P. Maes, "Concepts and experiments in computational reflection," in *Proceedings of the ACM Conference on Object-Oriented Languages (OOPSLA)*, pp. 147–155, ACM Press, December 1987.

[31] B. Redmond and V. Cahill, "Supporting unanticipated dynamic adaptation of application behaviour," in *Proceedings of the 16th European Conference on Object-Oriented Programming*, June 2002.

[32] R. Friedman and E. Hadad, "Client side enhancements using portable interceptors," in *Proceedings of the Sixth IEEE International Workshop on Object-oriented Real-time Dependable Systems*, January 2001.

[33] E. Wohlstadter, S. Jackson, and P. Devanbu, "DADO: enhancing middleware to support crosscutting features in distributed, heterogeneous systems," in *Proceedings of the International Conference on Software Engineering*, (Portland, Oregon), pp. 174–186, May 2003.