RAPIDware: Component-Based Development of Adaptive and Dependable Middleware

(www.cse.msu.edu/rapidware)

Philip. K. McKinley, R. E. Kurt Stirewalt, Betty H. C. Cheng,
Laura K. Dillon and Sandeep Kulkarni

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

August 2005

Abstract

Society depends increasingly on cyberinfrastructure for some of its most important functions. However, today's cyberinfrastructure is brittle and insecure, due in large part to the quality of the underlying software. A robust cyberinfrastructure must be able to adapt to changing conditions and protect itself from component failures and security attacks. Moreover, the design of such software systems should be grounded in rigorous software engineering techniques to assure their integrity and security under extreme conditions. This paper overviews the RAPIDware project, which investigates the design of high-assurance adaptive middleware, a critical step toward a robust cyberinfrastructure. Contributions of the project are reviewed, and experimental case studies are described.

Keywords: adaptive software, compositional adaptation, middleware, pervasive computing, autonomic computing, computational reflection, separation of concerns, component-based design, aspect-oriented programming, object-oriented programming, contract-based design, safe adaptation.

1 Introduction

Computing technology now affects nearly every dimension of modern society: managing critical infrastructure such as power grids and telecommunication networks; supporting electronic commerce and medical information systems; and controlling the operation of aircraft and automobiles. In addition, the "wireless edge" of the Internet continues to expand its reach, not only to support communication among mobile users, but also in the form of sensor networks for monitoring the physical environment. The increasing complexity of computing systems, and their interaction with the physical world, gives rise to the need for systems capable of self-management. *Autonomic computing* [1] refers to systems that are able to adapt to changing conditions, compensate for hardware and software failures, fend off attacks, and optimize performance, all with minimal human intervention. This capability is especially important to systems such as defense systems and communication networks, which must continue to operate correctly during exceptional situations. Such systems require run-time adaptation, including the ability to modify and replace components, in order to survive hardware component failures and security attacks.

We say a software application is *adaptable* if it can change its behavior dynamically (at run time) in response to transient changes in its execution environment (*e.g.,* to address dynamic network conditions) or to permanent changes in its requirements (*e.g.,* to upgrade long-running mission-critical systems). Developing and maintaining adaptable software are nontrivial tasks. An adaptable application comprises functional code, which implements the imperative behavior of the application (sometimes referred to as the *business logic*), and adaptive code, which implements the adaptive behavior of the application. The difficulty in developing and maintaining adaptable applications comes from the nature of the adaptive code, which tends to *crosscut* the functional code. Example crosscutting concerns include quality-of-service (QoS), fault tolerance, security, and energy management. Even more challenging than developing new adaptable applications is enhancing *existing* applications, such that they execute effectively in dynamic environments not envisioned during their original design and development. For example, many non-adaptive applications are being ported to mobile computing environments where they require dynamic adaptation.

One approach to designing adaptive software systems focuses on *middleware*, the layer of services between applications and underlying operating systems and network protocols. Middleware typically executes within the address space of the application, and therefore it can be "customized" using information specific to the particular application. On the other hand, middleware is designed to operate transparently with respect to the business logic of the application. In recent years, numerous studies have addressed the issue of how middleware can adapt to dynamic, heterogeneous environments to better serve applications [2]. The adaptive functionality in such systems can encompass not only changes in program flow, but also run-time changes to the composition of the software itself. Dynamic recomposition is needed when resource limitations (for example, memory in small devices) restrict the number of responding components that can be deployed simultaneously, or when adding new behavior to deployed systems to accommodate unanticipated conditions or requirements (for example, detection of and response to a new security attack). However, support for dynamic adaptation exacerbates the problem of assuring system integrity and security. As the demand for adaptive systems increases, a pressing challenge to the research community is to build a foundation of development technologies and tools, grounded in rigorous software engineering, that will enable the design and maintenance of high-assurance adaptive software systems.

The RAPIDware project investigates three overlapping aspects of this problem. First, we explore ways to enhance existing applications with new adaptive behavior, effectively providing a migration path to hardening existing software infrastructure. We focus primarily on approaches that can be realized through generative programming techniques [3]. We evaluate these mechanisms through case studies in both wired and wireless network testbeds. Second, we investigate design technologies that support assurance in adaptation. Specific topics include contract-based software design, safe adaptation, and analysis and verification of adaptive software. Third, we investigate technologies needed to enable robust and secure software adaptation across a network-centric infrastructure. Key topics include design of secure communication protocols and use of machine learning techniques to support decision-making for adaptation. In the remainder of this paper, we provide an overview of the work in each of these areas.

2 Adaptive Middleware: Background and Challenges

The interest in dynamically adaptive software systems has increased significantly during the past decade, in part due to advances in three supporting technologies: computational reflection, separation of concerns, and component-based design. *Computational reflection* refers to the ability of a program to reason about, and possibly alter, its own behavior [4]. Reflection enables a system to "open up" its implementation details for such analysis without compromising portability or revealing parts unnecessarily. In this approach, the program contains one or more *meta* levels, which enable reconfiguration of the underlying *base*-level code. *Separation of concerns* [5] enables the separate development of an application's functional behavior and its

adaptive behavior involving crosscutting concerns (e.g., quality of service, fault tolerance, security). This separation simplifies development and maintenance, while promoting software reuse [6]. A widely used technique is aspect-oriented programming (AOP) [7], where the code implementing a crosscutting concern, called an *aspect*, is developed separately from other parts of the system and *woven* with the business logic at compile- or run-time. Finally, *component-based design* [8] enables different parts of the system to be developed independently, potentially by different parties. Popular component-based platforms include COM/DCOM, .NET, CCM, and EJB. Compatible service clients and providers are coupled at run time through a process known as *late binding*. Component-based design facilitates reuse of software by enabling the assembly of off-the-shelf components provided by different vendors. Moreover, maintaining the component structure of a program after the initial deployment, when combined with late binding, facilitates dynamic recomposition [9].

With these technologies in hand, there are different ways to implement adaptive programs. One is to extend programming languages with explicit constructs for adaptation and reconfiguration. Open Java [10], R-Java [11], Handi-Wrap [12], and PCL [13] are examples. While this approach is an effective means to develop new adaptive programs, the developer needs to learn and understand the language extensions. Moreover, adding adaptive behavior to an *existing* non-adaptive program requires modifying the program source code directly. In other words, this approach is well suited to the development of new adaptable applications, but cannot be applied transparently to existing ones. An alternative approach is to use *middleware*, which provides a means to implement adaptation separately from the business logic of the application. Many adaptive middleware approaches are based on an object-oriented programming paradigm and are extensions of popular object-oriented middleware platforms such as CORBA, Java RMI, and DCOM/.NET. These approaches work by intercepting and modifying messages passing through the middleware. Examples include TAO [14], DynamicTAO [15], Open ORB [16], QuO [17], Squirrel [18], and IRL [19]. Other approaches implement adaptive behavior by extending a virtual machine with facilities to intercept and redirect interactions in the functional code. Examples of extensions to the Java Virtual Machine (JVM) include Iguana/J [20], metaXa [21] (previously called Meta Java), Guaraná [22], PROSE [23], and R-Java [11]. In general, approaches in this category are very flexible with respect to dynamic reconfiguration, in that new code can be introduced to the application at run time. However, while these methods provide transparency with respect to the application source code, they are applicable only to programs that are written for a specific middleware platform or virtual machine, thus limiting their applicability.

In the RAPIDware project, we focus on three complementary issues in the design of adaptive middleware. First, recognizing that many parts of the cyberinfrastructure already exist and will persist for many years, we investigate technologies that enable the *migration* from existing systems to those that are adaptive. A key research issue is how to do this in a manner that is both transparent to the application business logic and avoids dependencies on the underlying platforms. Second, we investigate techniques to improve the assurance of both new and existing adaptive middleware. The design of reconfigurable software must be supported by a programming paradigm that lends itself to *automated* checking of both functional and nonfunctional properties of the system [24]. In RAPIDware, we are developing tools and methods that apply rigorous software engineering to the entire software lifecycle of an adaptive system, from requirements, to design, to code generation, to run-time execution. Third, while middleware plays a key role in adaptive systems, adaptation often involves coordination across multiple system layers and among different platforms. Therefore, we are investigating techniques to facilitate such interoperation across different parts of the cyberinfrastructure.

3 Software Adaptation Mechanisms

At the core of most approaches to software adaptation is a level of indirection, with respect to object references and method invocations, that enables interception and redirection of interactions among program entities. An example is the use of a software *proxy*, whereby method calls to a target object must pass through a surrogate object that might redirect the calls to a different object. This indirection enables software components and control flow to be dynamically reconfigured, or replaced entirely, in response to changes in the execution environment [2].

Transparent Shaping. In RAPIDware we are exploring a variety of ways to enhance existing programs with new adaptive behavior. Collectively, these techniques comprise a new programming model, called *transparent shaping* [25–27], that enables new adaptive behavior to be woven into existing programs without modifying the application source code. Integration of the adaptive code is a two-phase process that combines aspect-oriented programming and computational reflection. In the first phase, the application is transformed into a new *adapt-ready* application by weaving "hooks" into the application code as needed. In the second phase, usually executed at run-time, an "adaptation infrastructure" is inserted dynamically using these hooks. Although this process incurs a (one-time) overhead, our experiments show the effect on the application is small. Once in place, the adaptation infrastructure supports insertion and removal of specific software sensors and actuators, enabling run-time adaptation and response to specific conditions.

We have designed and implemented transparent shaping tools for several commonly used programming languages and middleware platforms. First, we developed *Adaptive Java* [28], an extension to Java that contains constructs to support dynamic recomposition, and *MetaSockets* [29], an adaptable communication substrate for distributed applications. Second, we improved Adaptive Java by developing transparent reflective aspect programming (TRAP), which enables new adaptive behavior to be incorporated into existing programs automatically using code generation techniques. We have developed TRAP instances for Java [27] and for C++ [30]; a version for C# is under development. Third, we developed the Adaptive Corba Template (ACT) [31], which enables transparent shaping within existing middleware frameworks. Currently, we are exploring the relationship between adaptive systems and Parnas' program families [32, 33]. Specifically, we are investigating how transparent shaping can be used to generate one member of a *family* of adaptable programs given another program in the same family [34]. Placing these transformations under a formal umbrella enables reasoning about both software migration paths and dynamic reconfiguration during execution.

Experimental Case Studies. We have used transparent shaping to realize dynamic adaptation in existing applications. Several of our case studies address challenges arising in mobile computing settings: balancing quality-of-service and energy consumption on wearable and handheld computers [35–38], supporting autonomic handoff and QoS adaptation in heterogeneous multi-cell wireless networks [26, 27, 39], and dynamic configuration of "transient proxies" to enhance audio/video streaming across mobile ad hoc networks [40]. Other case studies focus on issues related to application integration: seamless interoperation of otherwise incompatible adaptive middleware frameworks [31], and development of composite Internet applications by (transparently) enabling them to interact using Web Services [41]. Recently, we proposed Service Clouds [42], a distributed infrastructure designed to facilitate rapid prototyping and deployment of autonomic communication services on the Internet. We have implemented a prototype of Service Clouds atop the PlanetLab Internet testbed [43] and used it to construct services for fast bulk data transfer and fault-tolerant multimedia conferencing.

4 Assurance in Adaptation

A major issue in the use of adaptive software mechanisms is *assurance*, namely how to specify, analyze, test, or otherwise verify that a system will always exhibit safety, liveness, and quality-of-service properties when it is deployed. In the RAPIDware project, we are investigating three main approaches to this problem: contract-based design; techniques to guarantee safe adaptation; and formal methods to specify and analyze adaptive systems.

Contract-Based Design of Adaptive Systems. Adaptive logic can often be separated from an application's business logic, where the separated concern is not completely orthogonal to the business logic. In RAPIDware, we are investigating two approaches to reasoning about application functionality when the logic for one or more concerns has been separated. The first and most mature approach extends Meyer's *design by contract* method [44] for reasoning about the interaction of application functionality and the separated concern. Beugnard and others [45] previously extended Meyer's approach from static specifications that are used at design time to entities that can be renegotiated at run-time as conditions change. We have developed a model of *synchronization contracts* [46], which declare an application's resource-usage patterns and which are negotiated by dynamically granting exclusive access to shared resources [46,47]. As an active process cycles through its synchronization states, resources are automatically acquired and/or released in order to keep the process in compliance with its contract. When an active process enters a state whose resource needs cannot be satisfied (e.g., because other processes already hold the needed resources) the process blocks until these resources become available. Our underlying middleware uses deadlock- and starvation-avoidance algorithms to fully automate negotiation of contracts, thereby separating synchronization and scheduling concerns from the business logic. We have used synchronization contracts to guard against serialization vulnerabilities, which pose serious security risks to multi-threaded applications [46, 48, 49]. Our second, more recent approach, exploits recent developments in formally-specified architecture description languages (ADLs) such as Wright [50] and connector wrappers [51], to reason about the orthogonality of concerns, such as reliability. We are using connector specifications to guide the development of a common-services middleware framework called Theseus, which supports highly asynchronous distributed systems [52, 53].

Safe Adaptation. A key issue when adding new adaptive behavior to an existing system is to ensure that a given adaptive action does not put the system into an inconsistent state. We are exploring two complementary approaches to addressing this problem. The first approach uses a distributed configuration management technique [54, 55], where dependency analysis is used to determine which software elements are affected by the candidate adaptive component. Using this dependency information, a safe adaptation graph (SAG) is constructed depicting all of the safe intermediate configurations. This SAG depicts more than one sequence of intermediate actions that can yield the intended adaptive action. As such, costs can be associated with various links, thus enabling optimization decisions to be made for selecting the most cost-effective approach to achieving a given adaptation. Once a request for adaptation is made, the software execution is directed to an appropriate safe point, at which the adaptation is made. If, during the adaptation process, an unexpected event or error occurs, then the system execution is rolled back to a point prior to the beginning of the adaptation process. The second approach uses a state-based representation for the intermediate states for a given adaptation. Specifically, a distributed application is modeled as a state-transition system and safe sequences of intermediate adaptive actions during adaptation process are identified [56]. A *transitional invariant lattice* is used to model the invariants that should be satisfied by the behavior of the intermediate adaptive actions. Once a safe sequence is identified, a program is instrumented to guarantee local adaptations do not occur out of order.

Specification and Analysis of Adaptive Systems. Adaptive systems typically adapt by adding, removing or replacing components. Component-based adaptation [2] allows for separation of concerns and independent development of components, and is essential towards building autonomic systems. As discussed earlier, transparent shaping offers adaptive mechanisms to support adaptation within program families (at the *implementation level*). Moving up one level of abstraction to the *design level*, we describe a *component family* as comprising components that provide similar functionality. Thus, depending upon the environment conditions and system requirements, one component may be replaced by another component in the family. In designing such a family of components, we separate *adapt-active* parts of each component from its functionality, thereby simplifying the specification and verification of adaptation. Only this adapt-active part is responsible for ensuring the correctness during adaptation [57]. We have found that the design of component families also helps in independent development of components and enhances reusability of components. Moreover, the use of the transitional-invariant lattice can be used to ensure that the adaptations between components in a family are indeed performed correctly [56].

At an even more abstract level, we have also introduced an approach for formally specifying adaptation *requirements* [58] in temporal logic [59]. We introduce A-LTL , an adaptation-based extension to linear temporal logic, and we use this logic to specify three commonly used adaptation semantics. We introduce adaptation semantics graphs to visually present the adaptation semantics. Specifications for adaptive systems can be automatically generated from adaptation semantics graphs. Leveraging this specification capability and using the separation of concerns approach for modeling adapt-active and functional portions of the system, we have developed a systematic process for specifying and analyzing the requirements of adaptive systems, including global system invariants [60]. We also support the automatic generation of executable prototypes to validate the requirements. We have applied this approach to a broad range of state-based modeling languages, including Petri nets, LOTOS, UML state diagrams and Z. Once the requirements have been validated and analyzed, then we can refine the specifications into components for design-level specification and verification, and eventually refine the specifications into code using transparent shaping code generation techniques.

5 Toward an Adaptive Network-Centric Infrastructure

Adaptation is not confined to middleware. An adaptive cyberinfrastructure requires interaction among applications, middleware, operating systems and networks, as well as coordination of adaptive behavior across different physical platforms. Moreover, it is imperative that these adaptive operations are carried out in a secure manner across the set of participating processes. In addition, the means by which the system decides how and when to adapt must be robust and capable of learning from past experience. In RAPIDware, we are addressing all three issues.

Distributed Collaborative Adaptation. Although middleware is very effective in realizing many types of adaptive behavior, others require cooperation among multiple system layers (vertical adaptation) and among distributed platforms (horizontal adaptation). We have investigated several techniques and protocols to support such interactions, including efficient monitoring protocols for overlay networks [61,62], an informed mobility protocol for managing energy in mobile ad hoc networks [63], cross-layer cooperation between middleware and the operating system kernel [40], and techniques to balance QoS and energy consumption in wireless networks [37]. The Service Clouds framework [42], mentioned earlier, supports vertical and horizontal adaptation by using overlay networks as a vehicle for distributed coordination. The framework is designed to be extensible: a suite of low-level UAF services for local and remote interactions can be used to construct higher-level adaptation services.

Secure Group Communication. Adaptability is also desired in network-centric group communication, where a group of users are collaborating on a common task. Encryption of the data is necessary to secure it from unauthorized access. However, to protect the security of the current users when the group membership changes, new keys must be distributed to users. We first developed a family of algorithms that enables the tradeoff between critical cost (the messages/encryptions ratio) and overall cost of rekeying [64]. Next, we investigated tradeoffs between storage cost and rekeying cost for secure multicast [65]. We developed a family of algorithms that provide a tradeoff between the number of keys maintained by users and the time required for rekeying due to revocation of multiple users. We showed that these methods can reduce the cost of rekeying by $43\% - 79\%$ when compared with previous solutions. We are currently evaluating these protocols using existing protocol analyzers [66, 67] to formally verify the adaptive properties. Finally, we investigated the problem of distributing key updates in secure dynamic groups [68] and proposed algorithms that reduce bandwidth by up to 55% when compared with previous algorithms.

Decision Making. In addition to software adaptation mechanisms and approaches to facilitating their correct and secure operation, an autonomic system also requires some means of *deciding* when conditions warrant a responsive adaptive action. Such decisions can be extremely complex, as software must process input not only on the state of its execution environment but, with the advent of pervasive computing, must also assess relative importance of different input from the physical world. In order for systems to "learn" from past experience and respond to unexpected situations, they must be able to filter an enormous number of input that may affect the decision. Moreover, many systems must make decisions in real time to prevent damage or loss of service. We are pursuing a two-pronged approach to the investigation of decision making for adaptive systems. The first approach explores how feature interaction analysis techniques can be applied to the decision-making process for adaptation [69]. The second approach uses statistical learning algorithms, previously used to help robots learn tasks, to aid decision-making in adaptive software [70].

Conclusions

As the complexity of computing systems increases, research in distributed autonomic computing is essential to the design of future systems that are robust, secure, adaptive and self-managing. Such software is likely to play a critical role in a variety of applications related to national defense and public safety. The RAPIDware project addresses several key aspects of high-assurance adaptive systems: adaptation mechanisms, assurance techniques, and network-centric infrastructure issues. Our ongoing investigations focus on the integration of these methods to support a wide variety of autonomic distributed systems. Moreover, this experimental research, combined with case studies involving industrial partners, will facilitate technology transfer and help to maximize the impact of this research on the state of the practice.

Further Information. Related papers and technical reports of the Software Engineering and Network Systems Laboratory can be found at the SENS website: `http://www.cse.msu.edu/sens`. Additional details on the RAPIDware project, including papers and software downloads, can be found at: `http://www.cse.msu.edu/rapidware`.

Acknowledgements. The RAPIDware project is supported in part by the U.S. Department of the Navy, Office of Naval Research under Grant No. N00014-01-1-0744. The authors would like to acknowledge the contributions of several students and other researchers: Reimer Behrends, Bru Bezawada, Karun Biyani, Eric Kasten, Masoud Sadjadi, Jesse Sowell, Chiping Tang, Zhenxiao Yang, Ji Zhang, Zhinan Zhou, Matthew Wallace, Jesus Bisbal, Peng Ge, Farshad Samimi, Mahesh Arumugam, Ali Ebnenasir, and David Knoester.

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] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng, "Composing adaptive software," *IEEE Computer*, vol. 37, no. 7, pp. 56–64, 2004.

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

[4] 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.

[5] P. Tarr and H. Ossher, eds., *Workshop on Advanced Separation of Concerns in Software Engineering at ICSE 2001 (W17)*, May 2001.

[6] K. J. Lieberherr, *Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns*. PWS Publishing Company, Boston, 1996. ISBN 0-534-94602-X.

[7] 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.

[8] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 1999.

[9] I. Ben-Shaul, O. Holder, and B. Lavva, "Dynamic adaptation and deployment of distributed components in Hadas," *IEEE Transactions on Software Engineering*, vol. 27, no. 9, pp. 769–787, 2001.

[10] M. Tatsubori, S. Chiba, K. Itano, and M.-O. Killijian, "OpenJava: A class-based macro system for Java," in *Proceedings of OORaSE*, pp. 117–133, 1999.

[11] J. de Oliveira Guimarães, "Reflection for statically typed languages," in *Proceedings of 12th European Conference on Object-Oriented Programming (ECOOP'98)*, pp. 440–461, 1998.

[12] J. Baker and W. Hsieh, "Runtime aspect weaving through metaprogramming," in *Proceedings of the first International Conference on Aspect-Oriented Software Development*, (Enschede, The Netherlands), 2002.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] R. Baldoni, C. Marchetti, and 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.

[20] 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.

[21] M. Golm and J. Kleinoder, "metaXa and the future of reflection," in *Proceedings of Workshop on Reflective Programming in C++ and Java*, pp. 1–5, 1998.

[22] A. Oliva and L. E. Buzato, "The implementation of Guaraná on Java," Tech. Rep. IC-98-32, Universidade Estadual de Campinas, Sept. 1998.

[23] A. Popovici, T. Gross, and G. Alonso, "Dynamic homogenous AOP with PROSE," tech. rep., Department of Computer Science, Federal Institute of Technology, Zurich, 2001.

[24] N. Venkatasubramanian, "Safe 'composability' of middleware services," *Communications of the ACM*, vol. 45, pp. 49–52, June 2002.

[25] Z. Yang, B. H. C. Cheng, R. E. K. 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 Systems (WOSS02)*, (Charleston, South Carolina), November 2002.

[26] S. M. Sadjadi and P. K. McKinley, "Transparent self-optimization in existing CORBA applications," in *Proceedings of the International Conference on Autonomic Computing (ICAC-04)*, (New York, NY), May 2004.

[27] S. M. Sadjadi, P. K. McKinley, B. H. C. Cheng, and R. E. K. Stirewalt, "TRAP/J: Transparent generation of adaptable java programs," in *Proceedings of the 2004 International Symposium on Distributed Objects and Applications*, (Agia Napa, Cyprus), October 2004.

[28] 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.

[29] 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.

[30] S. Fleming, B. H. C. Cheng, R. E. K. Stirewalt, and P. K. McKinley, "An approach to implementing dynamic adaptation in c++," in *Proceedings of the ICSE Workshop on Design and Evolution of Autonomic Application Software (DEAS)*, (St. Louis, Missouri), May 2005.

[31] 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)*, (Tokyo, Japan), March 2004.

[32] D. L. Parnas, "On the design and development of program families," *IEEE Transactions on Software Engineering*, March 1976.

[33] D. L. Parnas, P. C. Clements, and D. M. Weiss, "The modular structure of complex systems," in *Proceedings of the 7th International Conference on Software engineering*, pp. 408–417, 1984.

[34] S. M. Sadjadi, P. K. McKinley, and B. H. C. Cheng, "Transparent shaping of existing software to support pervasive and autonomic computing," in *Proceedings of the ICSE Workshop on Design and Evolution of Autonomic Application Software (DEAS)*, (St. Louis, Missouri), May 2005.

[35] P. K. McKinley, E. P. Kasten, S. M. Sadjadi, and Z. Zhou, "Realizing multi-dimensional software adaptation," in *Proceedings of the ACM Workshop on Self-Healing, Adaptive and self-MANaged Systems (SHAMAN), held in conjunction with the 16th Annual ACM International Conference on Supercomputing*, (New York City), June 2002.

[36] P. K. McKinley, S. Sadjadi, E. P. Kasten, and R. Kalaskar, "Programming language support for adaptable wearable computing," in *Proceedings of the Sixth International Symposium on Wearable Computers*, (Seattle, Washington), October 2002.

[37] Z. Zhou, P. K. McKinley, and S. M. Sadjadi, "On quality-of-service and energy consumption tradeoffs in FEC-enabled audio streaming," in *Proceedings of the 12th IEEE International Workshop on Quality of Service (IWQoS 2004)*, (Montreal, Canada), June 2004. selected as Best Student Paper).

[38] Z. Yang, Z. Zhou, B. H. C. Cheng, and P. K. McKinley, "Enabling collaborative adaptation across legacy components," in *Proceedings of the Third Workshop on Reflective and Adaptive Middleware (with Middleware'04)*, (Toronto, Ontario, Canada), October 2004.

[39] S. M. Sadjadi, P. K. McKinley, R. E. K. Stirewalt, and B. H. C. Cheng, "Generation of self-optimizing wireless network applications," in *Proceedings of the International Conference on Autonomic Computing (ICAC-04)*, (New York, NY), May 2004.

[40] F. Samimi, P. K. McKinley, S. M. Sadjadi, and P. Ge, "Kernel-middleware interaction to support adaptation in pervasive computing environments," in *Proceedings of the Second International Workshop on Middleware for Pervasive and Ad-Hoc Computing (with Middleware'04*, (Toronto, Ontario, Canada), October 2004.

[41] S. M. Sadjadi and P. K. McKinley, "Using transparent shaping and web services to support self-management of composite systems," in *Proceedings of the Second IEEE International Conference on Autonomic Computing (ICAC)*, (Seattle, Washington), June 2005.

[42] P. K. McKinley, F. A. Samimi, J. K. Shapiro, and C. Tang, "Service Clouds: A distributed infrastructure for composing autonomic communication services," Tech. Rep. MSU-CSE-05-31, Department of Computer Science, Michigan State University, East Lansing, Michigan, November 2005.

[43] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A Blueprint for Introducing Disruptive Technology into the Internet," in *Proceedings of HotNets–I*, (Princeton, New Jersey), October 2002.

[44] B. Meyer, *Object-Oriented Software Construction*. Prentice Hall, 1997.

[45] A. Beugnard *et al.*, "Making components contract aware," *IEEE Computer*, vol. 32, pp. 38–45, July 1999.
[46] R. Behrends, R. E. K. Stirewalt, and L. K. Dillon, "Avoiding serialization vulnerabilities through the use of synchronization contracts," in *Proc. of the Workshop on Specification and Automated Processing of Security Requirements*, pp. 207–219, Austrian Computer Society, Sept. 2004.
[47] R. Behrends, *Designing and Implementing a Model of Synchronization Contracts in Object-Oriented Languages*. PhD thesis, Michigan State University, East Lansing, Michigan USA, Dec. 2003.
[48] R. Behrends, R. E. K. Stirewalt, and L. K. Dillon, "A component-oriented model for the design of safe multi-threaded applications," in *Proceedings of the 8th International SIGSOFT Symposium on Component-based Software Engineering (CBSE 2005): Software Components at Work*, May 2005.
[49] R. E. K. Stirewalt, R. Behrends, and L. K. Dillon, "Safe and reliable use of concurrency in multi-threaded shared memory sytems," in *Proceedings of the 29th Annual IEEE/NASA Software Engineering Workshop*, 2005.
[50] R. Allen and D. Garlan, "A formal basis for architectural connection," *ACM Transactions on Software Engineering and Methodology*, vol. 6, pp. 213–248, July 1997.
[51] B. Spitznagel and D. Garlan, "A Compositional Formalization of Connector Wrappers," in *Proceedings of the 2003 International Conference on Software Eng ineering*, (Portland, Oregon, USA), May 2003.
[52] J. H. Sowell and R. E. K. Stirewalt, "Middleware reliability implementations and connector wrappers," in *Proc. of the ICSE Workshop on Architecting Dependable Systems (WADS'04)*, May 2004.
[53] J. H. Sowell and R. E. K. Stirewalt, "A feature-oriented alternative to implementing reliability connector wrappers," in *Architecting Dependable Systems III* (R. de Lemos, C. Gacek, and A. Romanovsky, eds.), Springer, 2005.
[54] J. Zhang, Z. Yang, B. H. C. Cheng, and P. K. McKinley, "Adding safeness to dynamic adaptation techniques," in *Proceedings of the ICSE 2004 Workshop on Architecting Dependable Systems*, (Edinburgh, Scotland), May 2004.
[55] J. Zhang, B. H. C. Cheng, Z. Yang, and P. K. McKinley, "Enabling safe dynamic component-based software adaptation," in *Architecting Dependable Systems III, Springer Lecture Notes for Computer Science* (A. R. Rogerio de Lemos, Cristina Gacek, ed.), Springer-Verlag, 2005.
[56] S. Kulkarni and K. Biyani, "Correctness of component-based adaptation," in *Proceedings of the International Symposium on Component-based Software Engineering*, May 2004.
[57] K. N. Biyani and S. S. Kulkarni, "Building component families to support adaptation," in *Proceedings of the ICSE Workshop on Design and Evolution of Autonomic Application Software (DEAS)*, (St. Louis, Missouri), May 2005.
[58] D. M. Berry, B. H. C. Cheng, and J. Zhang, "The four levels of requirements engineering for and in dynamic adaptive systems," in *11th International Workshop on Requirements Engineering Foundation for Software Quality (REFSQ)*, (Porto, Portugal), June 2005.
[59] J. Zhang and B. H. C. Cheng, "Specifying adaptation semantics," in *Proceedings of the IEEE ICSE Workshop on Architecting Dependable Systems (WADS)*, (St. Louis, Missouri), IEEE, May 2005.
[60] J. Zhang and B. H. C. Cheng, "Model-based development of dynamically adaptive software," in *IEEE International Conference on Software Engineering (ICSE06)*, (Shanghai, China), IEEE, May 2006. (accepted to appear).
[61] C. Tang and P. K. McKinley, "On the cost-quality tradeoff in topology-aware overlay path probing," in *Proceedings of the 11th IEEE International Conference on Network Protocols (ICNP)*, (Atlanta, Georgia), pp. 268–279, November 2003.
[62] C. Tang and P. K. McKinley, "A distributed approach to topology-aware overlay path monitoring," in *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS)*, (Tokyo, Japan), March 2004.
[63] C. Tang and P. K. McKinley, "iMobif: An informed mobility framework for energy optimization in wireless ad hoc networks," in *Proceedings of the Second International Workshop on Wireless Ad Hoc Networking (WWAN), in conjunction with the 25th IEEE International Conference on Distributed Computing Systems*, (Columbus, Ohio), June 2005.
[64] S. S. Kulkarni and B. Bruhadeshwar, "Adaptive rekeying for secure multicast," *IEEE/IEICE Special Issue on Communications: Transactions on Communications*, vol. E86-B, pp. 2948–2956, October 2003.
[65] S. S. Kulkarni and B. Bruhadeshwar, "Rekeying and storage cost for multiple user revocation," in *Proceedings of The 12th Annual Network and Distributed System Security Symposium*, (San Diego, California), 2005.
[66] P. Syverson and C. Meadows, "A formal language for cryptographic protocol requirements," *Designs, Codes, and Cryptography*, vol. 7, pp. 27–59, January 1996.

[67] C. Meadows, P. Syverson, and I. Cervesato, "Formal specification and analysis of the Group Domain Of Interpretation Protocol using NPATRL and the NRL Protocol Analyzer," *Journal of Computer Security*, vol. 12, no. 6, pp. 893–931, 2004.

[68] S. S. Kulkarni and B. Bruhadeshwar, "Distributing key updates in secure dynamic groups," in *Proceedings of the First International Conference on Distributed Computing and Internet Technology*, vol. 3347 of *Lecture Notes in Computer Science*, (Bhubaneswar, India), Springer, December 2004.

[69] J. Bisbal and B. H. C. Cheng, "Resource-based approach to feature interaction in adaptive software," in *ACM SIGSOFT Workshop on Self-Managing Systems (WOSS04)*, November 2004.

[70] E. P. Kasten and P. K. McKinley, "Meso: Perceptual memory to support online learning in adaptive software," in *Proceedings of the 3rd International Conference on Development and Learning (ICDL'04)*, (La Jolla, California), October 2004.