Applying Digital Evolution to the Development of Self-Adaptive ULS Systems[*]
(Position Paper)

Philip K. McKinley, Betty H. C. Cheng and Charles A. Ofria
Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan 48824
{mckinley,chengb,ofria}@cse.msu.edu

Abstract

A key characteristic for ultra-large scale (ULS) software-intensive systems is the need to adapt at run time in response to changing environmental conditions. Given the scale, complexity, and heterogeneity of ULS elements, innovative, but rigorous software engineering techniques are needed to address the development and the evolution of these systems. The developer of self-adaptive ULS systems must anticipate how and when the software will need to adapt in the future, codify this behavior in decision-making components to govern the adaptation, and ensure that system integrity is not compromised during adaptations. We contend that the full potential of dynamically adaptive software systems cannot be realized without environments that enable the developer to actively explore the "adaptation space" of the system during the early stages of design. We propose an approach to this problem that leverages and extends digital evolution techniques. By mapping models of adaptive software programs into digital organisms and studying traces of their evolution, the developer can gain critical insight into software decision making, software assurance, and the software infrastructure needed to support desired adaptations.

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. The pervasiveness of computing technology, coupled with its rapidly increasing complexity, gives rise to the need for computer 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. *Autonomic computing* [1] refers to systems capable of such self-management. This technology is especially important as computing systems interact increasingly with the physical world. For example, an increasing number of distributed applications involve data gleaned through wireless sensors that monitor and report various aspects of the physical environment. Since many of these applications involve public safety and national defense, enhancing their ability to detect and respond to changing conditions, including and potential threats, is paramount.

In the past decade, extensive research has been conducted on many aspects of self-adaptive software systems. Examples include adaptive software mechanisms [2], design of software to facilitate later modifications [3]; software-architecture techniques for supporting dynamic adaptation [4]; systems that distill data streams and distribute processing tasks to support resource-limited devices [5]; adaptable and extensible operating systems [6]; and dynamic recomposition within the network infrastructure itself [7]. This research has greatly improved our understanding of adaptive software and several key supporting concepts, including computational reflection, separation of concerns, component-based design, and transparent interception of program flow.

Despite these advances, however, designing an adaptive software system remains a very challenging task, due to the dynamic nature of adaptive software and uncertainty in the execution environment. We speculate that this problem is due in part to the fact that adaptive software is designed

[*]Research supporting the views stated in this position paper have been supported in part by NSF grants EIA-0000433, EIA-0130724, CDA-9700732, CCR-9901017, Department of the Navy, Office of Naval Research under Grant No. N00014-01-1-0744, Eaton Corporation, Siemens Corporate Research, and a grant from Michigan State University's Quality Fund.

and implemented using tools and environments intended for the development of *non-adaptive* software. We contend that the full potential of dynamically adaptive software systems cannot be realized without fundamental advances in the corresponding development environments, which must enable developers to explicitly address those features that distinguish adaptive systems from non-adaptive systems. These issues include anticipating how the software may need to adapt in the future, ensuring that system integrity is not compromised by adaptation, and constructing decision-making software to govern the adaptation. In the same manner that the introduction of other development tools (e.g., compilers, code generators, simulators, programming paradigms and languages) have improved software development in the past, so do we envision that the proper tools can lead to a new paradigm that more effectively supports the development of *adaptive* software.

To design self-adaptive computational systems, one can take inspiration from nature. Living organisms have an amazing ability to adapt to a changing environment, both in the short term (phenotypic plasticity) and in the longer term (Darwinian evolution). Indeed, most complex organisms exhibit traits that seem desirable in self-adaptive software: *system monitoring* (senses, awareness); *short-term changes in priorities* (stress reactions, sleep); *system reconfiguration* (muscle growth, calluses); *self repair* (tissue healing); *intrusion detection/elimination* (immune systems); and *maintaining state through transitions* (gradual acclimation to a new environment). In the past few years, several approaches to adaptive software design have sought to *mimic the behavior* of natural organisms (biomimetics), for example, [8]. However, natural organisms were produced through an evolutionary design process over millions of years. While biomimetics attempts to imitate the results of this process, it fails to account for the many factors that led to the adaptive behavior.

We propose a fundamentally different approach based on *digital evolution* [9, 10], where a population of self-replicating computer programs exists in a user-defined computational environment and is subject to mutations and natural selection. These "digital organisms" are provided with limited resources that they must carefully balance if they are to survive. Over time, these organisms will evolve to optimize their usage and thrive if they are able. Unlike mere numerical simulations or other forms of evolutionary computation (such as genetic algorithms), digital organisms possess the ability to *truly evolve*; the evolution is open-ended, often yielding unexpected solutions. No matter how well a given organism can perform a specific task, whether or not it self-replicates and moves into the next generation depends on its environment and its interaction with other organisms, not on an explicitly stated fitness function.

We are conducting preliminary investigations for a project, ORCHID,[1] that explores how digital evolution techniques can be leveraged and extended to support the design of robust self-adaptive software. The project focuses primarily on early development activities, namely requirements engineering and design, where we believe improvements in the early development processes can have the most profound effect on the quality and capabilities of future systems. Specifically, by mapping adaptive software systems to digital organisms and observing their evolution, the developer is able to actively explore the "adaptation space" of the system during the initial design. In this position paper, we provide background on digital evolution, describe the ways in which we believe digital evolution can be applied to the development of adaptive systems, and discuss some of the technical challenges that must be addressed to make this approach viable.

2 Digital Evolution Background

Digital evolution platforms have been developed first and foremost to provide a better understanding of evolution in nature [11]. Studying digital organisms offers several advantages over studying natural systems. As John Maynard Smith once noted,"So far, we have been able to study only one evolving system and we cannot wait for interstellar flight to provide us with a second. If we want to discover generalizations about evolving systems, we will have to look at artificial ones." Digital organisms enable scientists to address questions that are impossible to study with organic life forms, for example, explicitly disabling given mutations and seeing how that constraint affects the population.

However, unlike simulations of evolution, digital evolution is real. No matter how sophisticated an organism is at performing a single task, its likelihood of moving into the next generation depends on its interaction with the environment and other organisms, as well as its own ability at self replication. There is no explicit fitness function as in genetic algorithms; tasks only play a role in triggering an organism's environmental interactions. Like other forms of evolutionary computation, digital organisms can be used to design solutions to computational problems, where it is difficult to write explicit programs that produce the desired behavior [11]. Thus, digital organisms enable biological concepts to be used to study problems outside of biology, just as principles of physics and mathematics are often used in biology.

Currently, the most widely used digital evolution platform is AVIDA [9], developed by Ofria and colleagues.

[1]The project name was selected due to the evolutionary and adaptive nature of orchids, which are dated at being more than 90,000,000 years old, with 20,000 to 30,000 species, living on all continents except Antarctica.

Each Avida organism comprises a circular list of assembly-like instructions (its genome) and a virtual CPU, which executes those instructions. Residing in a common virtual environment, organisms may communicate with each other via the exchange of messages, resources may be produced and consumed, and organisms may sense and change properties of the environment. At any point in time the population of digital organisms may contain many different genomes. Some may be closely related (e.g., parent and offspring), while others may be related only through a distant ancestor. Each population starts with a single organism that is only capable of replication, and different genomes are produced through random mutations that occur during replication. Organisms are rewarded with virtual CPU cycles for performing specified *tasks*; selective pressure thereby produces organisms that exhibit beneficial behaviors and other traits. Tasks are generally defined in terms of externally visible behaviors of the organisms (their phenotype), rather than in terms of the specific instructions that must be executed by the digital organism's CPU. This approach allows maximum flexibility in the evolution of a solution for a particular task. Tracing the evolution of the AVIDA organisms can provide insight into a variety of problems, often revealing strikingly elegant solutions that the user did not expect.

Over the past several years, AVIDA has been used to conduct pioneering research in the evolution of biocomplexity. Specific studies address the evolutionary origin of complex traits [11], the evolutionary design of modularity [12, 13] and robustness [14, 15]. the evolution of multiple, interacting programs [10, 16, 17], the design of evolvable programming languages [18], and analysis techniques to breakdown the information contained within evolved code [19–21]. Indeed, the generality of AVIDA makes it an ideal platform to study questions associated not only with biology, but also with the application of biological concepts to software design.

3 Application to Software Development

Digital evolution can be used to support the design of robust self-adaptive systems in at least two ways. First, evolution can be used to discover robust communication protocols and strategies. In sensor networks, for example, communication operations typically require cooperation among many nodes in the network. Complex operations include well-studied issues in distributed computing, such as multicasting, gathering sensed data, leader selection, and detecting and responding to events of interest. Many traditional algorithms for solving these problems either perform poorly or are brittle when deployed in dynamic environments. On the other hand, digital evolution provides a means to explore a much larger solution space, potentially discovering algorithms that are more likely to remain mission-effective even under extremely harsh conditions. Our preliminary studies have demonstrated that AVIDA populations can evolve the ability to elect leaders [22] and forward data to a collection point [23], despite continuous turnover in the population. Algorithms produced in this manner can be codified and executed on real hardware.

The second way in which digital evolution can be used to help design self-adaptive systems is to use evolution of populations to help explore design of the software itself, in particular, models of its behavior in response to changing conditions. The ORCHID project is primarily about this second method, but also uses results of the first method to design adaptive software that can be deployed and tested in real-world environments. The software development tools and techniques produced by the ORCHID project will enable the designer to model software systems as digital organisms, observe their evolution under various conditions, and use the results to refine and improve the models. In related studies, McKinley, Cheng and colleagues have investigated several aspects of adaptive software design, In particular, the RAPIDware project [2] addresses high-assurance adaptive software, including programming language support [24,25], middleware support [26, 27], cross-layer cooperation distributed applications [28, 29], and techniques to maintain the state of the the system across adaptations [30, 31].

We argue that digital evolution can be used in concert with the above methods, by enabling the developer to explore, early in the design process, three aspects of self-adaptive systems that are particularly challenging using traditional software development tools: the decision making process for adaptation, preserving system integrity during reconfiguration, and designing the software infrastructure needed to realize adaptive behavior. Each area is discussed in turn.

Decision Making. In confronting a dynamic environment, a system may decide to adapt its behavior, and even modify its structure, to better fit the current environment. Indeed, many systems must make decisions in real time to prevent damage or loss of service. Decision makers use input gleaned from both the physical and the virtual environment in which they execute. A decision maker may have access to hundreds or thousands of sensor readings; the system must be able to capture the relative importance of different inputs, learn from past experience, and remember effective responses to the sensed environment. Many different techniques have been proposed to realize decision makers for adaptable software: rule-based methods, control theory models, resource optimization, emergent behavior, machine learning, and others.

Since the digital organisms have no pre-conceived notions about what the most optimal adaptive state will be, they will explore without regard for precedent, focusing only on what produces the best results for this specific case. In studies using AVIDA, digital organisms have shown a strong ability to produce novel behavior, which is often remarkably complex. We believe that applying them to systems such as these will enable us to find significant and unexpected adaptive pathways. Moreover, these studies have shown that digital organisms do have a remarkable ability to develop *phenotypic plasticity* in response to environmental inputs. In biology, this term refers to organisms that are genetically identical, but develop differently depending on what the environmental conditions are where they are living. We expect that similar adaptations during an organism's lifetime will be apparent in the digital organisms. One possible result of this study is the development of a hybrid approach to decision-making that subsumes rule-based, statistical, machine-learning, and control theory techniques. Such an approach might provide high-level mutations in the digital organisms that give the tools access to tools in these other areas. Those that utilize them most successfully will outcompete the rest of their population and be prime candidates for implementation as part of an adaptive system.

Safe Adaptation and State Management. Safe adaptation refers to preserving the integrity or consistency of the system as it adapts [31]. In many cases, recomposition of algorithmic or structural components at run-time requires the transfer of nontransient state information between an old component and its replacement. While the state capture problem has been addressed in other contexts, such as checkpointing, process or thread migration, mobile agents, the methods employed there generally are not directly applicable because they either incur too much overhead or do not support state transfer between different implementations of a component. Rather, dynamic software recomposition involves state transfer as it relates to *collateral change*, defined as the set of recompositions that must be applied to an application *atomically* for correct execution [30].

Digital organisms have demonstrated the ability to thrive in exceedingly complex environments with many different resources that interact with one another [11]. We have also witnessed their ability to group organismal traits together in changing environments to optimize their ability to survive. By a similar notion, we expect that these populations should inherently be able to handle large factor sets that must be dealt with atomically. The execution of many experiments will help the developer to analyze the particular choices of factors that the AVIDA organisms bundle together during evolution. Some of these connections occur by chance, but those that appear more frequently will likely have been motivated by selective pressure. It is these groupings that the developer must pay special attention to, so as to determine if their separation causes the system integrity to be compromised during adaptation.

Adaptive Infrastructure. The final issue we address in the design of adaptive systems is constraints on the degree to which the system is able to change its own behavior. As discussed by Kiczales [32] for meta-level interfaces in reflective systems, *preemption* occurs when the designer of an adaptive system makes a decision in the implementation that prevents a programmer (or another software entity) from using a feature of the system in a way that would otherwise seem natural. That is to say, decisions made when the framework is implemented preemptively restrict how the system can be adapted. Conversely, a completely *open* implementation implies that an application can be recomposed entirely at run-time. Hence, a major challenge for the designer of a self-adaptive system is to achieve the proper balance between preemption and openness in the system. This balance will be codified in a *adaptation infrastructure*, which should be flexible enough to support types of adaptation required of the system, yet should prevent a system from transforming itself in ways that are not desirable or threaten basic functionality of the system.

In this part of our study, we are investigating how digital evolution technologies can be leveraged to help the developer explore these issues during the early stages of design. Specifically, a candidate software design (architecture) can be represented as a digital organism. In contrast to the requirements modeling described above, this part of the project will focus on *design evolution*. As such, the digital organisms in this phase represent designs, and their environments are configured using the requirements for the collection of target systems in addition to the execution environment for the system. Preemption and openness are codified as mutations to the design organism, where tradeoffs will need to be made to best suit the environment of the target system requirements (behavior). Other constraints for the digital evolution process include the decision-making information and the assurance techniques. Our objective is to obtain one or more design organisms that optimally satisfy the set of target systems in the context of the relevant decision-making techniques and assurance constraints, while minimizing the required openness of the system.

4 Technical Challenges

While the use of digital evolution promises new capabilities to the designer of an adaptive system, we are just beginning our study of this approach. Several key technical issues need to be addressed in the full ORCHID project.

Requirements Engineering for Adaptive Systems. Most of the work on software engineering of adaptive systems has focused on "downstream" development issues, such as adaptation mechanisms [33], programming language support for adaptation, and software architecture support [34]. Relatively less research has addressed early development activities, such as requirements engineering (RE) and specification techniques for adaptive systems. There has been some preliminary work on specifying and verifying adaptive software [35–37] and on run-time monitoring of requirements conformance [38–40]. In addition, much of the work on *personalized* [41] and *customized* [42] software – at least with respect to eliciting, modeling, and reasoning about requirements variations – can also be applied to adaptive systems. To significantly improve assurance in dynamically adaptive software, more rigorous and systematic techniques for specifying, analyzing, and refining requirements for adaptive systems are needed. Berry *et al* [43] recently proposed a four-level model to represent the different types of RE taking place for and in dynamically adaptive systems. A key problem illuminated by this four-level model is the need for innovative techniques to identify the requirements and the corresponding behavioral models for target systems that can be adopted as part of the adaptation process. These target systems should represent the appropriate system behavior to handle different environmental conditions, including adverse conditions and possibly variations of these conditions.

Mapping Software Models to Digital Organisms. To address this issue, we are using digital evolution to create models for adaptive systems that might not be otherwise be discovered by human developers. We are investigating two main approaches. The first approach is to evolve AVIDA organisms that generate state diagrams during their execution, requiring minimal modifications to AVIDA. We demonstrated the feasibility of this approach in a preliminary study [44]. Specifically, we showed that digital evolution is capable of creating state diagrams describing wireless sensor systems. We were able to specify *what* properties the state diagram should satisfy, evolve organisms to generate diagrams, and use a model checker to *verify* that the diagram evolved by an organism adhere to the property.

In the second approach, a UML state diagram itself is encoded as the digital organism that evolves in response to changing environmental conditions. Using the state diagram as the digital organism is potentially more powerful than the method described above, however, a number of research challenges need to be addressed. First, mutation operators for a state diagram must be defined. These would likely include the addition and removal of states and transitions, but a more challenging aspect is formulation of the guards and actions. Guards and actions have a vocabulary that is system-dependent, so the evolving diagram must simultaneously evolve this vocabulary. This second approach has the distinct advantage that the of being immediately useful without any additional translation, and can potentially be set up with a more meaningful set of mutation operators where a single mutation would be able to add, remove, or duplicate states, or similarly modify transitions. Our plan is to use the results of our first approach to gain insight into how to introduce the mutations to guide a more optimized evolution of the models. For both cases, fitness criteria (codified as AVIDA tasks) will be defined to constrain the evolution process. Tasks can be defined to check whether diagrams satisfy system invariants; those that do will be rewarded and therefore be more likely to survive.

Representing Functional Requirements and Constraints. No matter which organism representation is used, the execution environment for the organisms must enforce all of the functional requirements and constraints of the system. Functional requirements are associated with resources that are placed in the environment for the organisms. Barely meeting a requirement will provide the organism with minimum of that specific resource needed for replication—an organism can only successfully replicate when it has met all of these requirements. Surpassing the minimum requirements will mean that the organism receives additional resources and will therefore be able to replicate more rapidly and, in turn, have a competitive advantage. Those requirements that are most important to optimize will be associated with the most valuable resources.

Constraints, on the other hand, can be thought of as "Laws of Physics." Example constraints include avoiding known feature interactions, dependencies among system requirements, and limitations on computational resources. We are exploring the most effective means to codify these constraints. Specifically, we need to assess the severity of these constraints to determine whether we want to discourage, hurt, or even terminate an organism when it violates one.

Mobile Computing Software Repositories. The initial target application domain of the ORCHID project is adaptive software for network-centric wireless applications, including mobile computing environments and wireless sensor networks. In order to facilitate the adaptation space exploration, we are leveraging our previous experience and that of others working in the area of adaptive software for mobile computing to identify commonly occurring patterns for requirements and design models. In particular, we need to express these patterns in terms of UML state diagrams for requirements and UML class diagrams for designs. Our previous experience with identifying requirements-level patterns for embedded systems [45] suggests that for a given domain, it is possible to identify recurring patterns for both requirements and designs. We are specifically looking for patterns in the same spirit as the design patterns by Gamma, where in addition to diagram templates, there are fields that describe the problem being addressed by a pattern, the consequences of using a pattern, constraints applicable to the pattern, example uses, dependencies among patterns, and suggested refinements to patterns for the next stage of development.

5 Summary

In summary, we propose that the field of digital evolution can be leveraged to improve the design of self-adaptive software for ULS systems. Specifically, development tools that model adaptive software as digital organisms, will empower the designers of self-adaptive systems with the ability to study the expected behavior of such systems, and detect potential problems, in ways not currently possible. The promise is that these techniques will yield software systems that are more robust and rigorous than those produced by existing methods.

Further Information. More information on MSU's Digital Evolution Laboratory and the AVIDA platform are available at http://devolab.cse.msu.edu. Additional information on the Software Engineering and Network Systems Laboratory and the RAPIDware project can be found at http://www.cse.msu.edu/sens and http://www.cse.msu.edu, respectively.

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] *Proceedings of the ACM SIGSOFT Workshop on Self-Healing Systems (WOSS02)*, (Charleston, South Carolina), November 2002.

[5] 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)*, (Heidelberg, Germany), pp. 252–272, November 2001.

[6] Y. Saito and B. Bershad, "System call support in an extensible operating system," *Software Practice and Experience*, vol. 1, pp. 1–10, January 1999.

[7] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A survey of active network research," *IEEE Communications Magazine*, vol. 35, pp. 80–86, January 1997.

[8] M. Wang and T. Suda, "The bio-networking architecture: A biologically inspired appro ach to the design of scalable, adaptive, and survivable/available network applic ations," Tech. Rep. 00-03, Department of Information and Computer Science, Unversity of C alifornia, Irvine, California, USA, February 2000.

[9] C. Ofria and C. O. Wilke, "Avida: A software platform for research in computational evolutionary biology," *Artificial Life*, vol. 10, pp. 191–229, 2004.

[10] S. S. Chow, C. O. Wilke, C. Ofria, R. E. Lenski, and C. Adami, "Adaptive radiation from resource competition in digital organisms," *Science*, vol. 305, pp. 84–86, July 2004.

[11] R. E. Lenski, C. Ofria, R. T. Pennock, and C. Adami, "The evolutionary origin of complex features," *Nature*, vol. 423, pp. 139–144, May 2003.

[12] C. Ofria and C. Adami, "Evolution of genetic organization in digital organisms," in *Proc. of DIMACS workshop Evolution as Computation*, (Princeton, NJ), pp. 167–175, 1999.

[13] D. Misevic, R. E. Lenski, and C. Ofria, "Sexual reproduction and muller's ratchet in digital organisms," in *Ninth International Conference on Artificial Life*, (Boston MA), pp. 340–345, 2004.

[14] R. E. Lenski, C. Ofria, T. C. Collier, and C. Adami, "Genome complexity, robustness and genetic interactions in digital organisms," *Nature*, vol. 400, pp. 661–664, 1999.

[15] C. O. Wilke, J. Wang, C. Ofria, C. Adami, and R. E. Lenski, "Evolution of digital organisms at high mutation rate leads to survival of the flattest," *Nature*, vol. 412, pp. 331–333, 2001.

[16] C. Ofria, C. Adami, T. C. Collier, and G. K. Hsu, "The evolution of differentiated expression patterns in digital organisms," *Lect. Notes Artif. Intell.*, vol. 1674, pp. 129–138, 1999.

[17] S. Goings, J. Clune, C. Ofria, and R. Pennock, "Kin selection: The rise and fall of kin-cheaters," in *Ninth International Conference on Artificial Life*, (Boston MA), pp. 303–308, 2004.

[18] C. Ofria, C. Adami, and T. C. Collier, "Design of evolvable computer languages," *IEEE Transactions in Evolutionary Computation*, vol. 17, pp. 528–532, 2002.

[19] C. Adami, C. Ofria, and T. C. Collier, "Evolution of biological complexity," *Proc. Natl. Acad. Sci. USA*, vol. 97, pp. 4463–4468, 2000.

[20] C. Ofria, C. Adami, and T. C. Collier, "Selective pressures on genomes in molecular evolution," *J. Theor. Biology*, vol. 222, pp. 477–483, 2003.

[21] W. Huang, C. Ofria, and E. Torng, "Measuring biological complexity in digital organisms," in *Proceedings of the Ninth International Conference on Artificial Life*, (Boston MA), pp. 315–321, September 2004.

[22] D. B.Knoester, P. K. McKinley, B. Beckmann, and C. A. Ofria, "Evolution of leader election in populations of self-replicating digital organisms," Tech. Rep. MSU-CSE-06-35, Department of Computer Science, Michigan State University, East Lansing, Michigan, December 2006.

[23] B. E. Beckmann, P. K. McKinley, D. B. Knoester, and C. A. Ofria, “Evolution of cooperative information gathering in self-replicating digital organisms,” Tech. Rep. MSU-CSE-07-10, Department of Computer Science, Michigan State University, East Lansing, Michigan, February 2007.

[24] 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.

[25] 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.

[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), pp. 88–95, May 2004.

[27] 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.

[28] 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).

[29] F. Samimi, P. K. McKinley, and S. M. Sadjadi, “Mobile service clouds: A self-managing infrastructure for autonomic mobile computing services,” in *Proceedings of the Second IEEE International Workshop on Self-Managed Networks, Systems and Services (SelfMan)*, (Dublin, Ireland), June 2006.

[30] E. P. Kasten and P. K. McKinley, “Perimorph: Run-time composition and state management for adaptive systems,” in *Proceedings of the 24nd International Conference on Distributed Computing Systems ICDCS’04*, (Tokyo, Japan), March 2004. to appear.

[31] 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. (in press).

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

[33] Z. Yang, B. H. 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 Software (WOSS’02)*, pp. 85–92, November 2002.

[34] J. P. Sousa and D. Garlan, “Aura: an architectural framework for user mobility in ubiquitous computing environments,” in *Proceedings of the third Working IEEE/IFIP Conference on Software Architecture*, pp. 29–43, 2002.

[35] S. Kulkarni and K. Biyani, “Correctness of component-based adaptation,” in *Proceedings of the International Symposium on Component-based Software Engineering*, May 2004.

[36] J. Zhang and B. H. C. Cheng, “Specifying adaptation semantics,” in *Proc. of the Work. on Architecting Depend. Sys. (WADS)*, pp. 1–7, 2005.

[37] Z. Zhou, J. Zhang, P. K. McKinley, and B. H. C. Cheng, “TA-LTL: Specifying adaptation timing properties in autonomic systems,” in *3rd IEEE Workshop on Engineering of Autonomic Systems (EASe 2006)*, (Columbia, Maryland), April 2006.

[38] S. Fickas and M. S. Feather, “Requirements monitoring in dynamic environments,” in *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, p. 140, IEEE Computer Society, 1995.

[39] W. N. Robinson, “Monitoring web service requirements,” in *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pp. 65–74, 2003.

[40] T. Savor and R. Seviora, “An approach to automatic detection of software failures in realtime systems,” in *Proc. IEEE Real-Time Tech. and Appl. Sym.*, pp. 136–147, 1997.

[41] A. Sutcliffe, S. Fickas, and M. M. Sohlberg, “PC-RE a method for personal and context requirements engineering with some experience,” *Req. Eng. J.*, vol. 11, no. 3, pp. 1–17, 2006.

[42] S. Liaskos, A. Lapouchnian, Y. Wang, Y. Yu, and S. Easterbrook, “Configuring common personal software: a requirements-driven approach,” in *RE ’05: Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE’05)*, (Washington, DC, USA), pp. 9–18, IEEE Computer Society, 2005.

[43] D. M. Berry, B. H. 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.

[44] H. Goldsby, D. Knoester, B. H. C. Cheng, P. McKinley, and C. Ofria, “Digitally evolving models for dynamically adaptive systems,” in *IEEE Proceedings of ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, (Minneapolis, Minnesota), May 2007. (to appear).

[45] S. Konrad, B. H. C. Cheng, and L. A. Campbell, “Object analysis patterns for embedded systems,” *IEEE Transactions on Software Engineering*, vol. 30, pp. 970–992, December 2004.