J Internet Serv Appl (2012) 3:51–58
DOI 10.1007/s13174-011-0049-4

SI: FOME - THE FUTURE OF MIDDLEWARE

Applying evolutionary computation to mitigate uncertainty in dynamically-adaptive, high-assurance middleware

Philip K. McKinley · Betty H.C. Cheng · Andres J. Ramirez · Adam C. Jensen

Received: 31 October 2011 / Accepted: 12 November 2011 / Published online: 3 December 2011

Abstract In this paper, we explore the integration of evolutionary computation into the development and run-time support of dynamically-adaptable, high-assurance middleware. The open-ended nature of the evolutionary process has been shown to discover novel solutions to complex engineering problems. In the case of high-assurance adaptive software, however, this search capability must be coupled with rigorous development tools and run-time support to ensure that the resulting systems behave in accordance with requirements. Early investigations are reviewed, and several challenging problems and possible research directions are discussed.

Keywords Middleware · Adaptive software · High-assurance · Evolutionary computation · Uncertainty

1 Introduction

Increasingly, computer systems need to adapt to their environment as they execute [1]. This need is most apparent in systems that interact with the physical world, such as sensor networks and swarms of autonomous robots. Such systems need to perform complex, distributed tasks despite lossy wireless communication, limited resources, and uncertainty in sensing the environment. However, even systems that operate entirely within cyberspace need to account for changing network and load conditions, component failures, and exposure to a wide variety of cyber attacks.

A *dynamically adaptive system* (DAS) monitors itself and its execution environment at run time. This monitoring enables a DAS not only to detect conditions warranting a reconfiguration, but also to determine when and how to *safely* reconfigure itself in order to deliver acceptable behavior before, during, and after adaptation [2]. Middleware has been shown to be an ideal location for adaptive functionality [3], since in many cases the adaptation can take place "below" the end application.

Within the context of a DAS, uncertainty arises from unanticipated environmental conditions as well as inaccuracy or imprecision of sensed values. Unfortunately, this uncertainty can compromise the decision-making capabilities of a DAS. Moreover, it is often infeasible for a human to know and/or enumerate all possible combinations of system and environmental conditions that a DAS may encounter [4]. As a result, automated methods are needed to explore this adaptation space, not only during execution, but also during the early stages of software development, where there is greater flexibility for resolving obstacles that prevent the satisfaction of goals.

Our research applies *evolutionary computation* (EC) to the design of high-assurance, self-adaptive software. EC methods, which codify the basic principles of genetic evolution in computer software [5], are particularly effective at producing solutions to problems with large, multidimensional search spaces. Indeed, the open-ended nature of the evolutionary process has been shown to discover novel solutions to complex engineering problems, rivaling and

P.K. McKinley (✉) · B.H.C. Cheng · A.J. Ramirez · A.C. Jensen
Department of Computer Science and Engineering, Michigan State University, 3115 Engineering Building, East Lansing, MI, USA
e-mail: mckinley@cse.msu.edu

B.H.C. Cheng
e-mail: chengb@cse.msu.edu

A.J. Ramirez
e-mail: ramir105@cse.msu.edu

A.C. Jensen
e-mail: acj@cse.msu.edu

even surpassing human designers [6]. In the case of high-assurance software, however, these techniques must be integrated with rigorous development tools and run-time support to ensure the resulting systems behaves in accordance with requirements.

In this paper, we discuss two main areas where evolutionary computation can support high-assurance adaptive systems: (1) Development tools and methods, including specification of requirements that explicitly address adaptation and environmental uncertainty, while identifying latent properties of software that could potentially lead to failures during run-time adaptation; and (2) Run-time support to enable adaptive monitoring of software requirements and environmental conditions, effective decision-making in the presence of uncertainty, and dynamic reconfiguration of software components. In each case, results of preliminary investigations are described, followed by discussion of the remaining challenges and possible directions for future research.

Many of our own research activities have been conducted in the context of robotics. Experiments are conducted on a testbed, called *Evolution Park*, depicted in Fig. 1. The testbed includes a heterogeneous swarm of mobile robots, parallel computing facilities for evolution runs and physics-based simulations, and a school of robotic fish in a large, custom-built water tank. In addition to the components shown in Fig. 1, a 3D printer enables fabrication of robot components that evolve concurrently with their control systems. This infrastructure supports evolution and experimental evaluation of both individual and cooperative behaviors, and is ideal for developing and testing the adaptive software technologies discussed in the remainder of this paper.

2 Background

The research community has responded to the demand for robust computational systems by producing a variety of technologies for developing high-assurance, self-adaptive software. Examples include adaptive software mechanisms [3, 7–9]; dynamic service composition [10], software-architecture-based techniques for supporting dynamic adaptation [11, 12]; and requirements-level and formal methods-based techniques [13–16].

Despite these advances, however, we are still far from producing software systems that exhibit adaptability, security, and survival "instincts" analogous to those found in natural organisms. Indeed, many researchers have sought inspiration from nature to help design resilient computational systems. One approach is *biomimetic* design, which takes a structure or behavior found in nature and attempts to replicate it in an artificial system.

Evolutionary computation An alternative is to harness the *process* that has produced robust behaviors in the natural world, evolution, and apply it to the development of adaptive computational systems. The most well-known EC method is the *genetic algorithm* (GA) [17], an iterative search technique in which the individuals in a population are encodings of candidate solutions to an optimization problem. In each generation, the most promising prospective solutions are selected and randomly mutated to create further diversity. Genetic programming (GP) [18] is a related method where the individuals are actual computer programs. Both GAs and GPs have been shown to be effective in a wide variety of science and engineering domains.

In contrast to these optimization techniques, *digital evolution* [19] is intended to explore the evolutionary process itself. In this method, self-replicating computer programs exist in a user-defined computational environment and are subject to mutations and natural selection. Over generations, these "digital organisms" can evolve to survive, and thrive, under dynamic and adverse conditions. The open-ended nature of digital evolution enables biologists to investigate fundamental questions that are difficult or impossible to study in natural systems [20]. Recently, digital evolution has also been shown to be effective in evolving robust communication protocols [21], in some cases revealing strikingly clever solutions that might otherwise not occur to human designers. This topic is discussed later.

Some EC approaches lie on the boundary between harnessing the power of natural selection and remaining tethered to biological principles. For example, neuroevolution [22] is a machine learning method in which GAs are used to train artificial neural networks (ANNs), which in turn can control systems such as robots. ANNs are particularly attractive for controllers because the inputs and outputs can directly correspond to sensors and actuators, respectively. In evolutionary robotics [23], an artificial genome encodes a robot's control system and possibly its morphology. The control program is downloaded into a real or simulated robot, which is then let "loose" in an environment. The fitness of the system is evaluated with respect to performing tasks. The most fit individuals in the population are allowed to reproduce, with random mutations and recombination, to form the next generation. This cycle is repeated until a satisfactory solution evolves.

Despite the contributions of EC to many fields, from physical design, to robotics, to biology, its potential in terms of constructing high-assurance software systems remains largely untapped. While natural systems have evolved to adapt to adverse conditions in remarkable ways, in order to produce similarly robust software that is also *assured*, EC methods must be coupled with rigorous software engineering methods. Such an integration will enable developers to exploit the tremendous search capability of EC while being able to ensure that essential properties of the system are

Fig. 1 Components of Evolution Park testbed: (*left*) collection of terrestrial microrobots and interactive simulation cluster; (*center*) rack-mounted parallel computing platform for evolution runs and behavior analysis; (*right*) custom-built tank for a school of robotic fish constructed from electroactive polymer materials

preserved across adaptations. In the following, we discuss several areas where EC can be applied to the development and run-time support of adaptive systems, pointing out directions for future research in this exciting area of study.

3 Harnessing evolution in adaptive software

Figure 2 illustrates the coupling of EC and the engineering of adaptive software. The center box lists the main topics addressed in this paper, where EC methods are integrated into both software development and run-time reconfiguration. Each is discussed below. These investigations are supported by a foundation of EC technologies, including traditional methods such as GA/GP [18], digital evolution [19], and neuroevolution [24]. The unique features of these different methods can be leveraged at different points in the life-cycle of dynamically-adaptive software.

As indicated by the box on the left, we advocate that such an approach should take advantage of traditional methods for developing high-confidence software, including (a) tools for analyzing evolved state diagrams, (b) model checkers for verifying adherence to critical system properties, and (c) methods from dynamical system theory to analyze continuous behavior and performance metrics. In addition, as indicated by the box on the right, these methods can directly integrate technologies developed specifically for adaptive software. Examples include the RELAX requirements-specification language, which enables uncertainty to be explicitly incorporated into specifications for dynamic adaptation; AMOEBA-RT, a run-time monitoring and verification technique for dynamically adaptive software; Transparent Shaping, which enables adaptive behavior to be woven into legacy code; and several formal modeling techniques for enabling safe adaptation.

The top of the Fig. 2 identifies several application domains where these technologies can be applied. Although the focus of our own research is primarily on robotics and autonomous systems, evolutionary computation can be integrated into design and run-time reconfiguration of a wide variety of high-assurance adaptive systems, either directly in application code or in supporting middleware platforms.

4 Development-time support

The first area we consider is the application of EC early in the software development process, "proactively" addressing anticipated dynamics and uncertainty in the execution environment. As DASs increase in complexity and interact with the physical world, it becomes impractical for a human to exhaustively explore the system and environmental conditions that may adversely affect operation of the system. EC methods can address this problem in the following ways.

Evolution-inspired algorithm design First, EC techniques can be used to evolve novel algorithms that are robust to adverse conditions. For example, we have conducted several studies using Avida [19], a digital evolution system developed originally for computational biology, but which we have extended to support distributed systems research. In Avida, each digital organism comprises a circular list of instructions (its genome) and a virtual CPU. Selective pressures representing desired high-level behaviors drive the evolutionary process. Organisms can sense the quantity of resources within the environment and perform computational tasks, either as individuals or as groups, to metabolize those resources into virtual energy. Organisms that are most successful—those that replicate faster, or make better use of resources—are more likely to spread and eventually dominate the population. Effectively, the Avida system provides software designers with a "digital Petri dish" for producing emergent computational behaviors that are able to

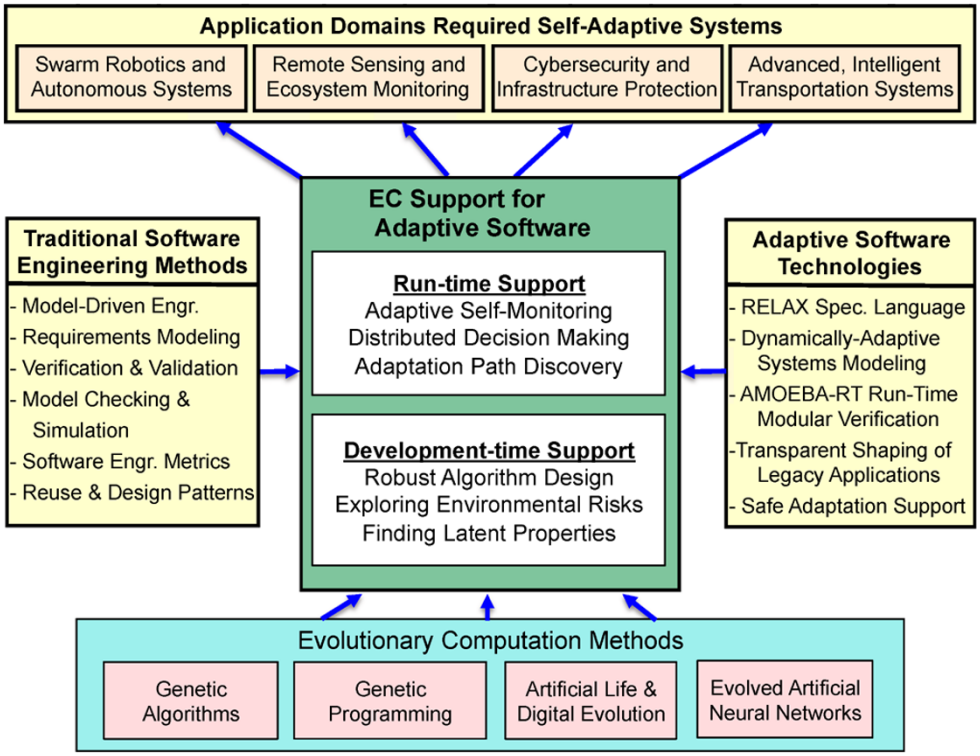

Fig. 2 Key areas of research involving the integration of evolutionary computation, model-driven software engineering, and high-assurance adaptable software. Shown *at the top of the figure* are several potential application domains

balance multiple conflicting concerns while operating under dynamic and adverse conditions.

Behaviors evolved in Avida, encoded in the genomes of organisms, can be cross-compiled and linked with an existing code base to create images suitable for execution. Previously, we demonstrated this "Avida-to-hardware" process by evolving a phototaxis behavior in Avida, automatically converting the code to C, and executing the program on iRobot Create robots. In addition, we can apply rigorous software development techniques to the evolving programs. For example, we can evaluate genomes using program analysis tools and reverse engineer genomes to formal specifications suitable for model checking.

We have applied Avida to evolve collective communication algorithms for a variety of distributed behaviors, including synchronization, quorum sensing, constructing networks, responding to attacks, and reaching consensus [21]. The evolved consensus algorithm [25] is particularly interesting because it employs a novel strategy based on probabilistic message forwarding. Effectively, the Avida program had "invented" a new, robust algorithm. The scope of future research in this area is nearly unbounded, as such techniques provide a means to discover solutions to complex problems in situations where human intuition may be limited, as well as to optimize those solutions to fit a target platform and environment.

Exploring environmental conditions In addition to applying evolutionary computation to *directly* develop solutions to complex problems, we can also harness its search capabilities to explore conditions a given system might encounter after deployment. Recently, Ramirez et al. developed LOKI [26], an automated technique for identifying combinations of system and environmental conditions that obstruct system requirements. LOKI leverages the concept of novelty search [27], a type of evolutionary algorithm where the fitness function is replaced by a domain-independent novelty function that measures the *difference* between solutions, rewarding solutions different from those previously discovered. Applied to a DAS, LOKI first uses a GA to generate sets of conditions. Through a simulation of the system, LOKI then exposes the DAS to these conditions and records how well the DAS satisfies requirements, as captured by a set of utility functions for requirements monitoring. Comparing the differences between values produced by these utility functions enables LOKI to evaluate the behavior of a DAS in response to perceived environmental conditions.

LOKI has previously been applied to an autonomous intelligent vehicle system (IVS) that performs adaptive cruise control, lane keeping, and collision avoidance. Experimental results demonstrated that LOKI was able to discover combinations of system and environmental conditions that lead to undesirable behaviors, such as requirements violations [26]. For instance, LOKI discovered a dangerous interaction that produced a requirements violation, where the IVS collided with a vehicle in front, temporarily departed from its driving lane due to the collision, and then repeatedly collided with

the same vehicle in order to reenter the driving lane. Such behaviors often correlated with high novelty values.

Discovering latent behavior While a requirements violation clearly obstructs a system requirement, complex systems may include *latent* behaviors, which are unexpected and potentially undesirable, but still satisfy requirements. Furthermore, certain unwanted behaviors may mean that requirements need to be modified to explicitly disallow the unwanted behavior. As a proof-of-concept, we developed an automated approach to identify temporal logic properties that describe latent, previously-unknown behaviors in existing UML models. A key component of our approach is an EC-based tool called Marple [28], which leverages natural selection to discover a set of properties covering different regions of the model state space. The discovered properties can be used to refine the models so as to remove unwanted behavior or to explicitly document a desirable property as a required system behavior.

Like LOKI, Marple uses novelty search, in this case to discover a set of properties that describe a UML model. Each individual within Marple represents a property created by instantiating commonly occurring specification patterns in the form of Linear Temporal Logic (LTL). Instantiating a pattern involves replacing the pattern's placeholders with evolved boolean propositions, where a proposition is created using attribute and operation information from a UML instance diagram of the system. Because the propositions can include logical conjunctives and disjunctives, the set of possible propositions is too large for brute force search methods to explore. During the evolutionary process, mutations and crossover produce different LTL properties that might be satisfied by the UML model. Next, the novelty of a property is assessed using the Spin model checker. If a novel region of the model state space is discovered, then the property is assigned a higher fitness value and Marple searches the new region more thoroughly. In this way, Marple discovers properties that cumulatively describe the behavior of the model. For validation, Marple was applied in an automotive industrial case study that identified a set of serious behavioral errors in a UML model describing four major electronic vehicle subsystems [29]. The errors had not been detected through traditional code generation and testing, and in the absence of the Marple tool, they would have been identified during integration testing only after a system prototype had been built.

Key challenges Despite promising results of our early investigations, several important problems need to be addressed to facilitate their adoption by the research and development communities. First, better tools are needed to bridge the gap between evolved solutions and software engineering methodologies. Specifically, while EC methods are capable of producing robust algorithms and protocols, translating these into forms amenable to formal analysis is often difficult. Techniques such as neuroevolution produce only weights associate with ANN links. Even when the evolved solution is *code*, as with Avida, the resulting program is produced by random mutations over evolutionary time. As a result, the encoding of behavior in the genome is often obscure, with code for different functions interwoven with "junk" code left over from earlier generations, making manual analysis tedious and time-consuming.

Second, one can envision the integration of LOKI-like functionality as an integral component of the design methodology. Specifically, the set of behaviors discovered by LOKI can be analyzed to identify (a) elements in the goal model that require new or augmented obstacle mitigations and (b) additional constraints to disallow undesirable latent behaviors. In the case of autonomous robots, for example, this information could be used to characterize environmental uncertainty and its effect on design considerations, such as physical placement of sensors, the level of redundancy in sensed information, and requirements to conserve energy.

Third, when considering the application of the proposed methods to physical systems, their integration with high-fidelity, physics-based simulators is essential. Only if the simulated environment accurately reflects the real world is it possible to use evolutionary computation to "predict" uncertainty that can arise due to the environmental conditions or malfunction of physical system components. Overcoming this "reality gap" is the subject of considerable research in the evolutionary robotics community.

Finally, thus far we have investigated these methods in stand-alone systems. To make this technology accessible to the development community, we and others need to integrate it into selected middleware platforms and test it on additional real-world applications.

5 Run-time monitoring and reconfiguration

While the above methods, applied at development time, can help to mitigate uncertain conditions eventually faced by the DAS, these techniques cannot address all eventualities. As it executes, a DAS needs to monitor itself and its environment, detect conditions warranting reconfiguration, determine which target system configuration will provide the desired behavior, and then select a series of reconfiguration instructions to reach that configuration. Collectively, these reconfiguration instructions form an *adaptation path*. To prevent loss of state or inconsistencies during a reconfiguration, a *safe* adaptation path preserves dependency relationships and ensures component communications are not interrupted [2]. Throughout this process, however, uncertainty in the sensed information poses a threat to the integrity of

the system. The second main area of EC-based research addresses how the DAS can accommodate uncertainty in self-monitoring and identify safe adaptation paths for reconfiguration.

Adaptive self-monitoring Self-monitoring inherently involves tradeoffs between the cost of monitoring and the accuracy, coverage, and coherence of gathered data. For example, a high monitoring rate may improve monitoring accuracy and data coherence, but it may also alter the behavior of a DAS in unpredictable and undesirable ways. Conversely, while a low monitoring rate can conserve system resources, it may also fail to detect events leading to a requirements violation.

We have previously developed Plato-RE [30], a requirements monitoring system that combines a genetic algorithm with utility functions to enable a DAS to detect possible requirements violations while minimizing resource consumption. Once utility functions detect a possible requirements violation, Plato-RE exploits the search capability of a GA to generate a new configuration that specifies the monitoring frequency for each component and sensor. We evaluated Plato-RE by using it to adapt the monitoring behavior of a simulated mobile robot. Experimental results show that Plato-RE can detect conditions conducive to a requirements violation while incurring lower monitoring overhead than the other adaptive monitoring approaches. This flexibility enables a DAS to automatically adapt the probing frequencies of given sensors in response to changing system and environmental conditions directly affecting the satisfaction of its requirements.

Decision making In a DAS, dynamic conditions can trigger actions intended to improve performance, conserve energy, or thwart an attack. However, optimizing one system concern may be in conflict with another (e.g., increasing connectivity in a mobile ad hoc network can improve network performance and fault tolerance, but could reduce the lifetime of the network by wasting energy). Moreover, such systems must be able to filter an enormous number of inputs that may affect the decision, which depends not only on the environment and platform capabilities, but also on the application domain and the specific mission being carried out. Recently, we have explored the application of EC to decision-making components of adaptive systems. In [31], we demonstrated the ability of a simple GA to support runtime decision making for a high-availability, distributed disk mirroring system. Instead of requiring developers to encode a predetermined set of adaptation strategies at design time to address anticipated reconfiguration scenarios, this system uses a GA to efficiently generate target reconfigurations that not only satisfy high-level directives provided by a system administrator, but also account for current system and environmental conditions.

We have also used neuroevolution to discover controller behavior for mobile autonomous agents [32], similar to nodes in a mobile ad hoc network (MANET). Specifically, we applied the NEAT system to solve a coverage problem, where nodes in a MANET were required to distribute themselves on a grid while maintaining network connectivity. The ANNs produced by NEAT were used as controllers for both the movement and communication behavior of nodes in the network, and all nodes in a given network executed a copy of the same evolved ANN. Nodes were provided with simulated radios, and were able to broadcast to their neighbors within a limited range. We found that approaches that implicitly reduced entropy, while explicitly addressing self-organization and scalability, were capable of discovering behaviors that remained stable even when controlling networks of different sizes than were evaluated during evolution. This result suggests that neuroevolution may be a viable strategy for discovering controllers for self-organizing multiagent systems.

Finding adaptation paths Once a dynamically adaptive system has detected conditions warranting reconfiguration and decided on a target system configuration, it must follow an adaptation path to reach that configuration. Although multiple safe adaptation paths may exist for a given situation, the identification and selection process is nontrivial, as different solutions may represent tradeoffs between reconfiguration costs, performance, and reliability.

Genetic programming can be used to automatically generate safe adaptation paths [33]. Instead of focusing on a single criterion when generating adaptation paths, this approach evolves solutions that balance competing objectives between functional and nonfunctional requirements, such as minimizing reconfiguration costs while maximizing reconfiguration performance and reliability. We developed a prototype system, Hermes, for this purpose. Each program evolved by Hermes comprises executable reconfiguration instructions that specify structural and behavioral changes a dynamically adaptive system must perform to safely reach a target reconfiguration. To facilitate the evolution of safe adaptation paths, Hermes is initialized with a set of *required* reconfiguration instructions derived by performing a component-dependency analysis between the current and target system configurations; all adaptation paths must minimally have these instructions. Then Hermes uses a GP to gradually transform and improve an adaptation path by adding, removing, replacing, and reordering reconfiguration instructions to better balance competing objectives, while safely reaching the desired target configuration.

Key challenges These initial studies provide a foundation for several exciting research directions. First is the possibility of executing a LOKI-like system at run time. In this

manner, the DAS could take into account uncertainty in the sensed data, combining sensor-related utility functions with self-modeling methods previously used in evolutionary robotics. This approach would enable the system to execute simulations of itself at run time and compare the results to the perceived environment, in order to make most effective use of its sensing capabilities while accounting for noise and faulty sensors. Such an approach could potentially be implemented within the context of requirements reflection [34], where a DAS is able to look inward during execution and assess its structure and behavior in the context of the system requirements.

Second, self-monitoring and decision making could be extended not only to consider issues such as cost and performance, but also to mitigate the impact of noise and uncertainty in the sensed environment. Like other factors, these issues could codified in utility subfunctions, such that the configurations found through evolutionary computation would exhibit redundancy to minimize their effect. If noise or a faulty sensor triggers an adaptive response from the system, then the DAS would continue to monitor conditions to determine if it adapted unnecessarily. If so, it would roll back to a previous state, and this experience should be incorporated into future actions.

Finally, the combination of these advances can potentially lead to the development of systems that are capable of on-board evolution after deployment. When such a system encounters unanticipated environmental conditions or component failures, it could evolve new strategies to mitigate adversity. The system might even maintain a population of "shadow" controllers evolved in different simulated environments, and which can be activated when similar situations are encountered.

6 Conclusions

In conclusion, evolutionary computation shows promise in supporting the development and run-time support for DASs. Examples include evolution of novel algorithms, characterizing disruptive environmental conditions, discovering potentially hazardous latent properties in software models, finding safe configurations and paths leading to them, and decision making in the presence of uncertainty. Encapsulating these capabilities into the middleware layer can reduce the effort required for a developer or application to make use of EC-based methods. Numerous exciting challenges lie ahead for the research community as we attempt to harness this power and integrate it with state-of-the-art software engineering technologies, ultimately enabling us to endow artificial systems with the robustness and resiliency of natural organisms.

Further information Studies in harnessing evolution for robust distributed computing and self-adaptive software, along with links to related materials, are described at: http://www.cse.msu.edu/thinktank. Information on the Evolution Park testbed is available at: http://www.cse.msu.edu/evopark.

Acknowledgements This work has been supported in part by NSF grants CNS-1059373, CNS-0915855, CNS-0751155, CCF-0541131, IIP-0700329, CCF-0750787, CCF-0820220, DBI-0939454, CNS-0854931, Army Research Office grant W911NF-08-1-0495, and Ford Motor Company. The authors gratefully acknowledge the other members of the Software Engineering and Network Systems Laboratory for their contributions to this work.

References

1. McKinley PK, Sadjadi SM, Kasten EP, Cheng BHC (2004) Composing adaptive software. IEEE Comput 37(7):56–64
2. Zhang J, Cheng BHC (2006) Model-based development of dynamically adaptive software. In: Proceedings of the 28th international conference on software engineering. ACM, New York, pp 371–380 (Distinguished Paper Award)
3. Blair GS, Coulson G, Robin P, Papathomas M (1998) 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
4. Whittle J, Sawyer P, Bencomo N, Cheng BHC, Bruel J-M (2009) RELAX: Incorporating uncertainty into the specification of self-adaptive systems. In: Proceedings of the 17th international requirements engineering conference (RE '09), Atlanta, Georgia, USA. IEEE Computer Society, Washington, pp 79–88
5. De Jong KA (2002) Evolutionary computation: a unified approach. MIT Press, Cambridge
6. Awards for human-competitive results produced by genetic and evolutionary computation. Competition held as part of the annual genetic and evolutionary computation conference (GECCO), sponsored by ACM SIGEVO. Results available at http://www.human-competitive.org
7. Schmidt DC, Levine DL, Mungee S (1998) The design of the TAO real-time object request broker. Comput Commun 21:294–324
8. Sadjadi SM (2004) Transparent shaping support for adaptability in pervasive and autonomic computing. PhD thesis, Michigan State University, East Lansing, Michigan, USA
9. Vanegas R, Zinky JA, Loyall JP, Karr DA, Schantz RE, Bakken DE (1998) QuO's runtime support for quality of service in distributed objects. In: Proceedings of the IFIP international conference on distributed systems platforms and open distributed processing (Middleware'98), The Lake District, England, September
10. Mokhtar SB, Georgantas N, Issarny V (2007) COCOA: COnversation-based service COmposition in pervAsive computing environments with QoS support. J Syst Softw 80(12):1941–1955
11. Kramer J, Magee J (2007) Self-managed systems: an architectural challenge. In: Future of software engineering 2007. IEEE-CS Press, Los Alamitos
12. Fleury F, Solberg A (2009) A domain specific modeling language supporting specification, simulation and execution of dynamic adaptive systems. In: Proceedings of the 2009 international conference on model driven engineering languages and systems (Models '09), Denver, Colorado, USA. Lecture notes in computer science, vol 5795. Springer, Berlin, pp 606–621

13. Fickas S, Feather MS (1995) Requirements monitoring in dynamic environments. In: Proceedings of the second IEEE international symposium on requirements engineering. IEEE Computer Society, Washington, p 140
14. Allen R, Douence R, Garlan D (1998) Specifying and analyzing dynamic software architectures. In: Proceedings of the 1998 conference on fundamental approaches to software engineering (FASE'98), Lisbon, Portugal, March
15. Kramer J, Magee J (1998) Analysing dynamic change in software architectures: a case study. In: Proc of 4th IEEE international conference on configurable distributed systems, Annapolis, May
16. Zhang J, Cheng BHC (2006) Model-based development of dynamically adaptive software. In: Proceedings of international conference on software engineering (ICSE'06), Shanghai, China, May
17. Holland JH (1975) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. University of Michigan Press, Ann Arbor
18. Koza JR (2003) Genetic programming IV: routine human-competitive machine intelligence. Kluwer Academic, Norwell
19. Ofria C, Wilke CO (2004) Avida: a software platform for research in computational evolutionary biology. J Artif Life 10:191–229
20. Lenski RE, Ofria C, Pennock RT, Adami C (2003) The evolutionary origin of complex features. Nature 423:139–144
21. McKinley PK, Cheng BHC, Ofria C, Knoester D, Beckmann B, Goldsby H (2008) Harnessing digital evolution. IEEE Comput. 41
22. Stanley KO, Miikkulainen R (2004) Competitive coevolution through evolutionary complexification. J Artif Intell Res 21:63–100
23. Floreano D, Husbands P, Nolfi S (2008) Evolutionary robotics. In: Handbook of robotics. Springer, Berlin
24. Miikkulainen R, Stanley KO (2008) Evolving neural networks. In: GECCO '08: proceedings of the 2008 GECCO conference companion on genetic and evolutionary computation. ACM, New York, pp 2829–2848
25. Knoester DB, McKinley PK (2009) Evolution of probabilistic consensus in digital organisms. In: Proceedings of the third IEEE international conference on self-adaptive and self-organizing systems, San Francisco, California, September
26. Ramirez AJ, Jensen AC, Cheng BHC, Knoester DB (2011) Automatically exploring how uncertainty impacts the behavior of dynamically adaptive systems. In: Proceedings of the 26th international conference on automated software engineering (ASE11), Lawrence, Kansas
27. Lehman J, Stanley KO (2008) Exploiting open-endedness to solve problems through the search for novelty. In: Proceedings of the eleventh international conference on artificial life (ALIFE XI). MIT Press, Cambridge
28. Goldsby HJ, Cheng BHC (2010) Automatically discovering properties that specify the latent behavior of UML models. In: Proceedings of the ACM/IEEE international conference on model driven engineering languages and systems (MoDELS 2010), Oslo, Norway, October
29. Jensen A, Cheng B, Goldsby H, Nelson E (2011) A toolchain for the detection of structural and behavioral latent system properties. In: Proceedings of the ACM/IEEE international conference on model driven engineering languages and systems
30. Ramirez AJ, Cheng BHC, McKinley PK (2010) Adaptive monitoring of software requirements. In: Proceedings of the first international workshop on requirements at run time, Sydney, Australia, October
31. Ramirez A, Knoester D, Cheng BHC, McKinley PK (2009) Applying genetic algorithms to decision making in autonomic computing systems. In: Proceedings of the 6th IEEE international conference on autonomic computing and communications, Barcelona, Spain, June. Best Paper Award
32. Knoester DB, McKinley PK (2011) Neuroevolution of controllers for self-organizing mobile ad hoc networks. In: Proceedings of the fifth IEEE international conference on self-adaptive and self-organizing systems, Ann Arbor, Michigan, October
33. Ramirez AJ, Cheng BHC, McKinley PK, Beckmann BE (2010) Automatically generating adaptive logic to balance non-functional tradeoffs during reconfiguration. In: Proceedings of the 7th international conference on autonomic computing, Washington, DC, June, pp 225–234
34. Bencomo N, Whittle J, Sawyer P, Finkelstein A, Letier E (2010) Requirements reflection: requirements as runtime entities. In: Proceedings of the 32nd ACM/IEEE international conference on software engineering, ICSE '10, vol 2. ACM, New York, pp 199–202