Automatically Generating Adaptive Logic to Balance Non-functional Tradeoffs During Reconfiguration

Andres J. Ramirez, Betty H.C. Cheng, Philip K. McKinley, and Benjamin E. Beckmann
Michigan State University
Department of Computer Science and Engineering
3115 Engineering Building
East Lansing, MI 48823
{ramir105, chengb, mckinley, beckma24}@cse.msu.edu

ABSTRACT

Increasingly, high-assurance software systems apply self-reconfiguration in order to satisfy changing functional and non-functional requirements. Most self-reconfiguration approaches identify a target system configuration to provide the desired system behavior, then apply a series of reconfiguration instructions to reach the desired target configuration. Collectively, these reconfiguration instructions define an adaptation path. Although multiple satisfying adaptation paths may exist, most self-reconfiguration approaches select adaptation paths based on a single criterion, such as minimizing reconfiguration cost. However, different adaptation paths may represent tradeoffs between reconfiguration costs and other criteria, such as performance and reliability. This paper introduces an evolutionary computation-based approach to automatically evolve adaptation paths that safely transition an executing system from its current configuration to its desired target configuration, while balancing tradeoffs between functional and non-functional requirements. The proposed approach can be applied both at design time to generate suites of adaptation paths, as well as at run time to evolve safe adaptation paths to handle changing system and environmental conditions. We demonstrate the effectiveness of this approach by applying it to the dynamic reconfiguration of a collection of remote data mirrors, with the goal of minimizing reconfiguration costs while maximizing reconfiguration performance and reliability.

Categories and Subject Descriptors

I.2.8 [**Computing Methodologies**]: Artificial Intelligence, Problem Solving, Control Methods, and Search

General Terms

Experimentation

Keywords

Autonomic computing, evolutionary algorithm, genetic programing, intelligent control, self-reconfiguration

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICAC'10, June 7–11, 2010, Washington, DC, USA.
Copyright 2010 ACM 978-1-4503-0074-2/10/06 ...$5.00.

1. INTRODUCTION

Increasingly, high-assurance software systems reconfigure themselves dynamically to satisfy functional and non-functional requirements as system and environmental conditions change [15]. A dynamically adaptive system detects conditions warranting reconfiguration, determines which target system configuration will provide the desired system behavior, and then selects and applies an adaptation path to reach the desired target configuration. An *adaptation path* comprises a series of reconfiguration steps. To prevent loss of state or introduction of erroneous results during a reconfiguration, a *safe* adaptation path preserves dependency relationships and ensures component communications are not interrupted [13, 14, 24]. Although multiple safe adaptation paths may exist for a given situation, the identification and selection process is non-trivial, as different solutions may represent tradeoffs between reconfiguration costs, performance, and reliability. This paper introduces an evolutionary computation-based approach that automatically generates adaptation paths to safely transition an executing system to its desired target configuration, while balancing tradeoffs between functional and non-functional requirements.

Most traditional self-adaptive systems apply variants of rule-based [3, 9, 24] and utility-based [2, 16, 22] techniques to select applicable reconfiguration plans at run time. These reconfiguration plans encode sequences of reconfiguration instructions that define an adaptation path. Kramer and Magee [13] introduced an approach for dynamically reconfiguring distributed systems while preserving system consistency. Zhang *et al.* [24] extended Kramer and Magee's approach by generating sets of safe adaptation paths and then selecting an adaptation path that minimized system disruption during reconfiguration. Viable adaptation paths, however, may represent complex tradeoffs between reconfiguration costs, performance, and reliability. Previously, Ramirez *et al.* [19] introduced Plato, an evolutionary computation-based approach for generating target system reconfigurations at run time. Plato, however, did not specify the sequence of reconfiguration instructions required to safely transition an executing system to a new target configuration.

This paper introduces Hermes, an evolutionary computation-based approach for automatically generating adaptation paths that safely transition an executing system from its current configuration to its desired target configuration. Hermes harnesses the process of evolution to

efficiently explore parts of a vast solution space comprising all possible adaptation paths. Moreover, instead of focusing on a single criterion when generating adaptation paths, Hermes evolves solutions that balance competing objectives between functional and non-functional requirements, such as minimizing reconfiguration costs while maximizing reconfiguration performance and reliability. Additionally, Hermes can be applied at design time to generate alternative adaptation paths, and at run time to generate safe adaptation paths that handle changing system and environmental conditions.

Hermes applies *genetic programming* to efficiently evolve safe adaptation paths. In contrast to many other evolutionary computation-based techniques, genetic programming [12] generates *executable* programs that solve specific and complex tasks, such as regression and robotic controllers. As such, 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 [13, 24] between the current and target system configurations; all adaptation paths must minimally have these instructions. Hermes then uses evolutionary techniques 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.

To demonstrate the effectiveness of this approach and leverage previous results on generating interesting target configurations [19], we applied Hermes to the dynamic reconfiguration of a network of remote data mirrors [8, 11] with the main objective of minimizing reconfiguration costs, while maximizing reconfiguration performance and reliability. Experimental results indicate that Hermes can significantly improve the overall quality of existing adaptation paths in real-time. The remainder of this paper is organized as follows. Section 2 provides background and related work on remote data mirroring, dynamic change management control, and genetic programming. In Section 3 we describe the design, implementation, and configuration of Hermes. In Section 4 we present the results of applying Hermes to an industrial-scale problem. We then analyze experimental results in Section 5. Lastly, we summarize our findings and briefly discuss future work in Section 6.

2. BACKGROUND & RELATED WORK

This section first presents the concept of remote data mirroring. We then overview dynamic change management and present related work. Lastly, we describe genetic programming and explain how it automatically generates executable programs.

2.1 Remote Data Mirroring

Remote data mirroring is a technique that stores copies of data at physically distinct locations, thereby isolating important data from failures that may affect either the original data or the duplicate copies at remote mirrors [8]. Remote data mirroring is a complex and expensive task that should be done only when the cost of losing data outweighs the cost of protecting it [11]. Essentially, remote data mirroring enables access to important data even if one copy is lost, corrupted, or becomes unreachable. For example, in the event of a failure, a remote data mirror may either reconstruct lost data or failover to another remote data mirror that contains the desired data. Remote data mirrors may either propagate data *synchronously*, where secondary sites receive and apply writes before the write completes at the primary, or *asynchronously*, where updates accumulate at the primary site and are periodically sent to secondary sites [11]. While it is often desirable to keep data as consistent as possible between remote data mirrors, remote data mirroring designs generally involve tradeoffs between performance, cost, and potential for data loss.

2.2 Dynamic Change Management

Developers often design and implement adaptation paths at design time to address specific reconfiguration scenarios that may arise at run time [2, 3, 9, 23]. For example, Figure 1 illustrates an adaptation path for reconfiguring the underlying topology in a network of remote data mirrors. In this setting, adaptation paths comprise a series of reconfiguration instructions that modify the structure and behavior of a system. Given a starting and a target system configuration, developers can either hand code or automatically generate these adaptation paths. Several of these automated approaches [21, 24] are based on Kramer and Magee's dynamic change management approach for reconfiguring distributed computing systems while preserving system consistency during the reconfiguration.

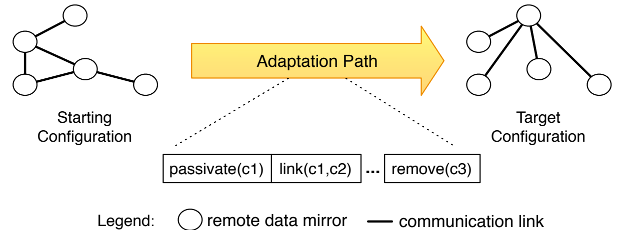

Figure 1: Reconfiguring a network of remote data mirrors.

In Kramer and Magee's dynamic change management model [13], components reach active, passive, and quiescent operational states in bounded time. *Active* components may initiate, accept, and service requests. For instance, an active remote data mirror may send requested data to other remote data mirrors while also diffusing new data to be replicated across the network. In contrast, *passive* components may accept and service requests, but may not initiate new requests nor be currently engaged in a transaction they initiated. A passive remote data mirror, for example, may send requested data to other remote data mirrors, but may not diffuse new data across the network. *Quiescent* components, on the other hand, are neither engaged in a transaction nor will they receive or initiate new requests. Thus, a quiescent remote data mirror will neither diffuse data across the network nor receive data requests from other remote data mirrors. To reach a quiescent state, however, all neighboring components must first reach a passive state that prevents components from receiving or initializing requests during reconfiguration.

Kramer and Magee's quiescence requirement for safe adaptation [13] may result in significant degradation of system performance because components not involved in an

adaptation may need to temporarily reach passive states. Vandewoude *et al.* [21] introduced the concept of *tranquility* as a weaker but sufficient condition for preserving system consistency during adaptation. In contrast to quiescence, tranquility does not require neighboring components to reach passive states before a component undergoes a reconfiguration. Specifically, only components involved in a reconfiguration must reach passive states, thus providing tranquility an advantage of being less disruptive than quiescence. However, Vandewoude *et al.* also showed that reaching tranquility in a bounded time is not guaranteed. For instance, a component undergoing adaptation may receive requests from active neighboring components at any time. As a result, reaching tranquility is considerably influenced not only by the order in which requests from neighboring components arrive, but also by whether those requests overlap or not. Although experimental evaluations conducted by Vandewoude *et al.* [21] showed this scenario to be rare, if tranquility is not reachable in bounded time, then the system must regress to a quiescent state.

Zhang *et al.* [24] further extended Kramer and Magee's approach by generating graphs of all possible safe adaptations. Each edge in the safe adaptation graph encodes a safe reconfiguration step. Furthermore, each reconfiguration step in the graph is associated with a relative cost that measures the approximate time required for the reconfiguration step to complete. Once the safe adaptation graph is generated, Zhang *et al.* apply Dijkstra's algorithm to search for a safe reconfiguration that minimizes system disruption. While their approach guarantees globally optimal solutions, the complexity of the algorithm is exponential with respect to the number of components involved in the reconfiguration. As a result, their approach may be impractical if reconfigurations involve many components.

Goldsby *et al.* [5, 6] applied digital evolution to evolve suites of structural and behavioral models of adaptive systems (i.e., target configurations) that not only satisfy functional requirements, but also provide different tradeoffs between non-functional properties. In addition, Ramirez *et al.* [18, 19] introduced a real-time evolutionary computation-based approach to explore system configurations that balance non-functional tradeoffs according to current system and environmental conditions. Although both approaches explore broader sets of target adaptive system configurations, neither approach produces the sequence of reconfiguration instructions required to safely transition the executing system from one target configuration to another.

2.3 Genetic Programming

Genetic programming is a search-based technique that generates executable programs to solve specific tasks [12]. In a genetic program, an individual encodes a candidate executable program in a genome comprising sets of instructions and terminals that can be executed directly on a computer or on a virtual environment; typically, a tree-based genome representation is used to facilitate a hierarchical evaluation of the program. In contrast, Hermes uses an *interpreted linear* genome representation (see Figure 2). Specifically, a genome in Hermes encodes an adaptation path comprising a vector of reconfiguration instructions. Each reconfiguration instruction can be mapped to low-level implementation code responsible for enacting structural and behavioral changes throughout the adaptive system. In addition, the position of an instruction in a linear representation explicitly determines the order in which it is executed, which may or may not be the case with tree-based program representations [1]. This explicit ordering of instructions is critical when evaluating the safety of an adaptation path.

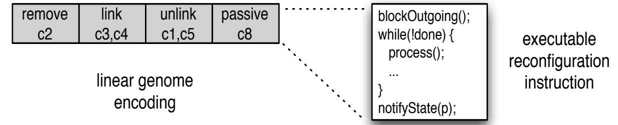

Figure 2: Linear-based program encoding.

Genetic programs examine multiple individuals in parallel, storing them in a population. Fitness functions are used to map the quality of an individual's encoded solution to a scalar value. A genetic program uses this fitness value to compare the relative qualities of different individuals and thus selects better solutions. To generate new solutions, genetic programming applies two key operators, crossover and mutation. The *crossover* operator exchanges genetic material from two existing individuals in the population to create two new individuals, each representing different solutions. As Figure 3 illustrates, in a linear representation, two-point crossover first randomly selects two individuals from the population as parents, A and B. Two indices are then randomly selected from each parent to indicate the range of instructions that will be exchanged. These instructions are then swapped, creating two new individuals, AB and BA. Ideally, the crossover operator exchanges key *building blocks*, or groups of instructions that have meaning in isolation, between existing individuals to generate more fit solutions as offspring [7].

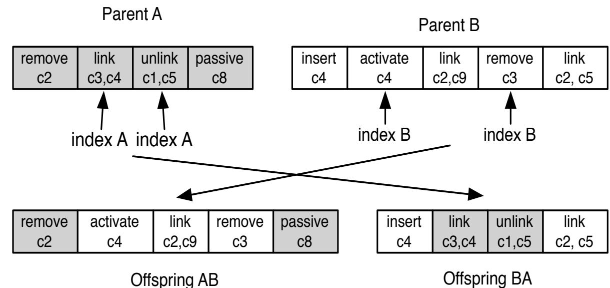

Figure 3: Two-point crossover in linear genetic program.

In contrast to the crossover operator, the *mutation* operator explores points in the solution space that perhaps are not currently found in the population [12]. The main objective of the mutation operator is to introduce variation into the population by randomly inserting, removing, and replacing instructions and terminals in randomly selected genomes [7]. Figure 4 illustrates an individual who is selected from the population and mutated. Specifically, in Figure 4, "*remove(c2)*" is replaced by "*insert(c4)*" and "*insert(c3)*" is inserted into the genome, thus producing A'. Ideally, the mutation operator introduces variation into the population that will form part of the overall solution. Genetic programs typically execute until either the maximum number of iterations or generations are exhausted, or a fitness value within some specific threshold is reached.

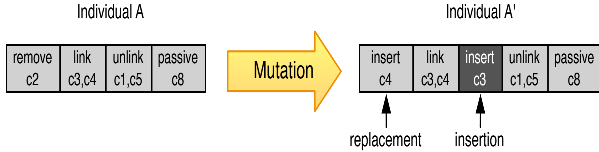

Figure 4: Mutation in linear-based genetic program.

3. PROPOSED APPROACH

Hermes generates safe adaptation paths between starting and target system configurations. Evolved solutions must satisfy the following two constraints, while balancing multiple, potentially competing, factors affecting the system. First, evolved adaptation paths may not reconfigure the executing system to configurations other than the specified target system. Second, evolved adaptation paths may never cause a dynamically adaptive system to reach an inconsistent or erroneous state. Therefore, if an adaptive system begins a reconfiguration in a consistent state, then it will also reach the target system configuration in a consistent state. We next describe how we applied Hermes to the dynamic reconfiguration of a remote data mirror network with the primary objective of minimizing reconfiguration costs while maximizing reconfiguration performance and reliability.

The proposed approach assumes the existence of an adaptive infrastructure comprising elements for monitoring [4, 17], decision-making [2, 10, 18, 19], and reconfiguration [3, 20, 23, 24] to support self-adaptation with assurance at run time. Additionally, it is assumed that the functional logic in the dynamically adaptive system implements a dynamic management interface that enables an adaptation driver to query and control the operational status of components towards *active*, *passive*, or *quiescent* states in bounded time. With this infrastructure available, Hermes accepts monitoring information, and starting and target system configurations as inputs, and then produces safe adaptation paths that balance non-functional requirements as output. As such, developers may leverage Hermes at design time to generate suites of adaptation paths that may then be encoded into an adaptive system. Furthermore, Hermes may also be leveraged at run time to select and apply the best evolved safe adaptation path based on current system conditions.

3.1 Design

The following steps describe Hermes's design and implementation for reconfiguring networks of remote data mirrors. Note that Hermes may be applied to other application domains by extending the instruction set with application-specific reconfiguration instructions.

Instruction Set. The core instruction set that Hermes uses to construct safe adaptation paths is derived from Kramer and Magee's dynamic change management approach [13]. These instructions, comprising primitive reconfiguration operators, are briefly described in Table 1. An adaptation driver can issue these instructions to control the operational status of system components and reconfigure the application at run time. In addition, each reconfiguration instruction in Hermes is associated with a specific cost measuring the approximate amount of time required for the instruction to complete. For this paper, cost values reflect approximate and relative estimates for the domain of remote data mirroring. In practice, these costs may be refined with empirical measurements gathered by the monitoring infrastructure.

Table 1: Description of reconfiguration instructions used by our genetic program.

Instruction	Description	Cost (s)
Insert Component	Adds component to the network.	10
Remove Component	Removes component from the network.	3
Link Components	Establishes a communication path between the specified components.	3
Unlink Components	Removes a communication path between the specified components.	2
Activate Component	Sets the operational status of a component to active mode.	1
Passivate Component	Sets the operational status of a component to passive mode.	5

Terminal Set. In genetic programming, a terminal set specifies which objects are arguments of instructions. The terminal set in Hermes comprises remote data mirrors (components), as well as network links (connectors) that can be established between pairs of remote data mirrors. The specific terminal set is derived at run time by analyzing structural differences between starting and target system configurations. Thus, for this application the terminal set only includes remote data mirrors and network links involved in the reconfiguration.

GP Operators. Hermes applies a two-point crossover operator, as previously illustrated in Figure 3. In addition to the basic insertion, removal, and replacement operators, the mutation process in Hermes also applies a *swap operator*. The swap operator randomly exchanges the locations of two instructions in the genome and thus explores the effects of executing reconfiguration instructions in different orders.

Selection. In evolutionary algorithms, selection is the process by which better solutions thrive and sometimes even dominate the population. Hermes applies *tournament selection*, a variation that randomly selects k individuals from the population and then competes them against each other. The individual with the highest fitness value *survives* into the next generation where it may undergo further recombination and mutation.

Initialization. Genetic programs must be properly configured for the specific task being solved. Table 2 lists several genetic program parameters along with the specific values used in Hermes. Although these values were effective in the remote data mirroring case study, developers should explore various parameters when applying Hermes to other application domains.

In addition to the common genetic program configurations, Hermes requires an additional setup step. While most genetic programs begin execution from a "blank slate" population comprising random individuals, Hermes initializes each individual's encoded program with an initial adapta-

Table 2: Genetic program configuration.

Parameter	Value
Population Size	1000
Crossover Type	Two-point
Crossover Rate	20%
Mutation Rate	50%
Selection Type	Tournament, $k = 5$
Selection Rate	30%
Max. Generations	1500

tion path comprising specific instructions *required* to safely transition a system to its target configuration. These instructions are derived through component-dependency analysis [13, 24]. For example, if two remote data mirrors are connected in the target configuration but not in the starting configuration, then it can be deduced that somewhere in the adaptation path, both remote data mirrors must be linked. To preserve these required reconfiguration instructions in the population, no mutation operator in Hermes may remove them. This additional constraint, which is not typical of genetic programming, is needed because the starting and ending points of the evolutionary process are known; traditional genetic programming is more open-ended, where the objective of evolution is finding interesting endpoints. In contrast, Hermes is looking for interesting paths to get to known endpoints (i.e., target configuration). As a result, Hermes may modify the initial adaptation path in any possible way as long as it safely reconfigures the adaptive system to its target configuration.

It is important to consider the complexity of the solution space comprising all possible adaptation paths. First, we define n to be the number of instructions required to safely transition the system to its target configuration, as determined by component-dependency analysis. Exactly $n!$ possible alternative solutions may be constructed by simply reordering the initial genome. Given that most non-trivial reconfigurations may comprise well over 20 instructions, the solution space comprises over 2.43×10^{18} different adaptation path alternatives, some of which safely transition the system to its desired target configuration, while many others do not. Given the vast number of combinations possible, no current method (manual or automated, heuristic or brute-force exhaustive) is capable of exploring all possibilities in a reasonable amount of time.

3.2 Fitness Sub-Functions

It is often the case that multiple fitness or utility functions are required to evaluate a single solution for multiple, and often competing, objectives [2]. For example, a particular remote data mirroring design that maximizes network performance is likely to either incur high operational costs or provide inadequate data protection. In general, most practical remote data mirroring solutions attempt to provide adequate data reliability and network performance without incurring excessive operational costs. A set of fitness sub-functions can be used to evaluate competing objectives, each focusing on a different concern. To this end, Hermes applies a set of fitness sub-functions derived and elaborated from published results in remote data mirroring [8, 11] and search-based software engineering [5, 24] domains.

Cost. The first criterion we consider in remote data mirroring is the cost of a reconfiguration. For this paper, we measure reconfiguration costs as the time required to execute an adaptation path. The following fitness sub-function measures the cost of reconfiguration:

$$F_{\text{cost}} = 100 - \left(\frac{\text{time}_{ev} - \text{time}_{init}}{\text{time}_{init}} * 100 \right)$$

where time_{init} and time_{ev} measure the amount of time required for the initial and evolved adaptation paths to complete, respectively. This fitness sub-function guides the selection of individuals whose encoded solution reconfigures the network of remote data mirrors in less time. While Hermes associates reconfiguration costs with the time required to complete a reconfiguration, this time measurement could also be further refined into lost profits due to system disruption.

Performance. Another important criterion in remote data mirroring is the performance degradation caused by a reconfiguration. For this paper, we determine the performance of a reconfiguration by measuring the amount of data produced and diffused by remote data mirrors through the network during reconfiguration. The following two fitness sub-functions measure the performance of an encoded solution:

$$F_{p_{act}} = 100 * \left(\frac{\text{components}_{act}}{\text{components}_{tot}} \right)$$

and

$$F_{\text{data}_{sent}} = \sum_{i=1}^{\text{components}_{tot}} \text{time}_{act}(i) * \text{capacity(i)}$$

where components_{act} is the number of components in active mode during reconfiguration, components_{tot} is the total number of components in the system, time_{act}(i) measures the time a remote data mirror i is actively diffusing data during reconfiguration, and capacity(i) measures the data output produced by a remote data mirror per time unit. The first performance fitness sub-function, $F_{p_{act}}$, measures the percentage of components in the system in active mode throughout the reconfiguration. The second performance fitness sub-function, $F_{\text{data}_{sent}}$, measures the amount of data diffused through the network by remote data mirrors during reconfiguration. Together, these two fitness sub-functions guide the genetic program towards solutions that maximize the number of components actively diffusing data through the network during reconfiguration.

Reliability. The third criterion we consider for remote data mirroring is the reliability, or potential for data loss, of a reconfiguration. For this paper, we determine the reliability of a reconfiguration by measuring the amount of data *queued* during reconfiguration. The following two fitness sub-functions measure the reliability of an encoded solution:

$$F_{r_{pass}} = 100 * \left(\frac{\text{components}_{pass}}{\text{components}_{tot}} \right)$$

and

$$F_{\text{data}_{q}} = \sum_{i=1}^{\text{components}_{tot}} \text{time}_{pass}(i) * \text{capacity(i)}$$

where components_{pass} is the number of components in passive mode during reconfiguration, components_{tot} and capacity(i) are the same as defined above, and time_{pass} measures the time a remote data mirror i is in passive mode

throughout the reconfiguration process. The first fitness sub-function, Fr_{pass}, measures the percentage of passivated remote data mirrors in the system. The second fitness sub-function, F_{data_q}, measures the amount of data produced by remote data mirrors that is queued because the remote data mirror was in passive mode at the time. Together, these two fitness sub-functions guide the genetic program towards solutions that create large regions of quiescence during reconfiguration. From the perspective of data reliability, establishing large regions of quiescence throughout the system is desirable because it implies data is better protected against failures during reconfiguration.

In Hermes, each set of fitness sub-functions is associated with a vector of coefficients that determines the relative priority of each design concern. By default, coefficients in this vector are all equivalent, implying that no one competing design objective is more significant than others. However, system requirements and environmental conditions may impose different constraints upon the type and quality of the evolved adaptation path. For instance, if the cost of losing data outweighs performance requirements, then the coefficient for reliability should be set to a value larger than that of performance and cost, thus guiding the evolutionary algorithms towards solutions that provide greater measures of reliability during reconfiguration. This vector of coefficients can also be updated at run time to address changing system and environmental conditions. Moreover, the set of fitness sub-functions and the vector of coefficients can be combined into a single scalar fitness value through a linear weighted sum, as follows:

$$\begin{aligned} \text{Fitness} &= \alpha_{\text{cost}} * F_{\text{cost}} + \alpha_{\text{perf}} * (Fp_{act} + F_{\text{data}_{sent}}) + \\ &\quad \alpha_{\text{rel}} * (Fr_{pass} + F_{\text{data}_q}) - \text{penalties} \end{aligned}$$

where *penalties* are reductions in fitness meant to punish individuals whose encoded solution produces undesirable effects or behaviors. For instance, any evolved adaptation path that either fails to transition the system to its desired target configuration, or does so while violating safety constraints, is severely penalized. The objective of penalizing individuals is to guide the evolutionary algorithm towards valid and meaningful solutions by removing individuals from the population with undesirable behaviors that do not promote safe adaptation.

4. EXPERIMENTAL RESULTS

This section presents a set of experiments conducted to evolve safe adaptation paths that reconfigure a network of remote data mirrors [11]. Each experiment compares the relative fitness value of adaptation paths evolved by Hermes with those derived by component-dependency analysis. Specifically, we compare our results with Kramer and Magee's dynamic change management algorithm [13]. We selected this particular algorithm because it generates safe adaptation paths and is scalable. While the algorithm presented by Zhang *et al.* [24] generates globally optimal solutions that minimize system disruption, the algorithm is not scalable for the input sizes considered in these experiments. Similarly, the tranquility approach introduced by Vandewoude *et al.* [21] may not be practical for this domain as remote data mirrors frequently propagate large amounts of data and bounded time adaptation is essential.

To compare our results, we implemented Kramer and Magee's dynamic change management protocol [13], which is shown in Algorithm 1. The input for this algorithm comprises a set of components to be inserted (N_c) and removed (N_r), a set of links to be created or removed (LS), and a set of components that must be passivated (CPS). The set of components to be passivated (CPS) comprises all components in N_r, all components with links to any component in N_r, and all components with a link in LS. The algorithm proceeds by passivating all components in CPS, thereby establishing a region of quiescence to preserve system consistency during reconfiguration. The algorithm then removes links from LS present in the system, followed by all components in N_r. Next, components in N_c are inserted, followed by creating all remaining links in LS. Finally, all inserted components, as well as those in CPS that were not removed, are set to active mode, thereby completing the reconfiguration process.

Algorithm 1 Reconfigure(N_c, N_r, LS, CPS)

```
for all i in CPS do
  Passivate i
end for

for all i in LS do
  if LS(i) exists then
    Unlink i
    LS = LS - LS(i)
  end if
end for

for all i in N_r do
  Remove i
end for

for all i in N_c do
  Create i
end for

for all i in LS do
  Link i
end for

for all i in {CPS − N_r + N_c} do
  Activate i
end for
```

Each experiment was run on a MacBook Pro with a 2.53GHz Intel Core 2 Duo Processor and 4GB of RAM. For each set of results presented in this section, we performed 100 trials and plot the mean along with corresponding standard error bars. Random starting and target system configurations were generated for each trial. Although the following experiments explore a wide range of possible reconfigurations, including the insertion and removal of several remote data mirrors at run time, each trial focuses mostly on reconfiguring the network topology. This decision was based on the observation that inserting and removing numerous remote data mirrors at run time is generally impractical due to excessive operational costs.

Finally, for the following experiments, we defined the *null* hypothesis, H_0, to state that *adaptation paths evolved by* Hermes *will show no difference in quality when compared to adaptation paths generated through component-dependency analysis*. Furthermore, we define the alternative hypothesis, H_1, to state that Hermes *will generate solutions better in*

quality than those produced through component dependency-analysis. For each experiment, the quality of two different adaptation paths is determined by comparing the fitness values associated with each adaptation path.

4.1 Base Comparison

This experiment compares the relative quality of adaptation paths evolved by Hermes with those obtained from component-dependency analysis. In this experiment we consider a typical scenario where the primary objective is to safely reconfigure the network of remote data mirrors while minimizing reconfiguration costs and maximizing reconfiguration performance and reliability, i.e., $\alpha_{cost} = \alpha_{perf} = \alpha_{rel} = 0.333$. In addition, we explore the performance characteristics of Hermes by applying our approach to networks of varying sizes and topologies, where a larger network size typically implies a more complex reconfiguration.

Figure 5 shows the average maximum fitness values for adaptation paths in this experiment. In particular, adaptation paths evolved by Hermes achieved greater fitness values than those produced by component-dependency analysis, with a statistical significance of $p < 0.01$ using a t-test. This difference in fitness values implies that adaptation paths generated by component-dependency analysis can be optimized by using Hermes to provide better reconfiguration performance and reliability with a minimal increase in reconfiguration costs. Furthermore, the difference in fitness values gradually increases as the networks and adaptation paths grow in size and complexity (n=15 and n=25). This observation suggests that more opportunities for balancing competing objectives arise as the complexity of a reconfiguration increases. This observation also suggests that an approach such as Hermes is capable of exploiting such opportunities to improve the overall quality of safe adaptation paths.

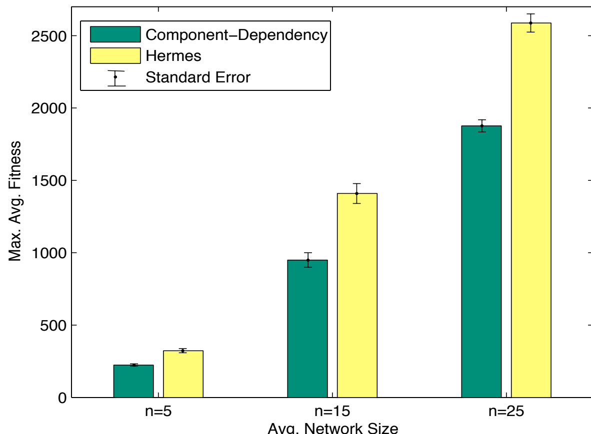

Figure 5: Comparison of adaptation path quality.

Figure 6 plots the average maximum fitness values of solutions evolved by Hermes for different sized networks per generation. This plot illustrates the rapid rate at which Hermes builds upon and improves the quality of adaptation paths generated by component-dependency analysis, which are represented by the fitness value plotted at the 0 generation before Hermes modifies them (filled icon). Specifically, Hermes achieves large boosts in fitness values within the first 600 generations (< 35 seconds), depending upon the relative size of the network. Thereafter, Hermes continues to fine-tune evolved adaptation paths until the maximum number of generations are exhausted.

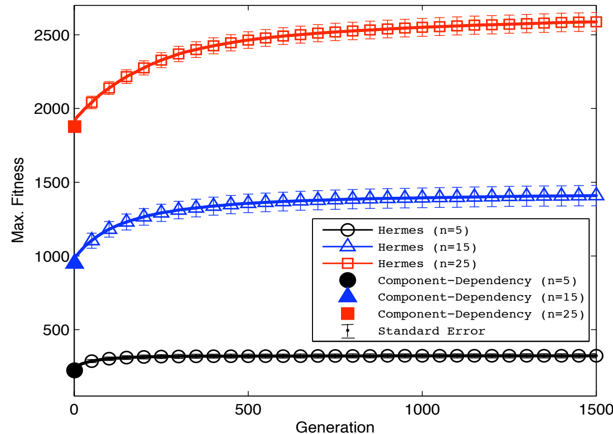

Figure 6: Progression of average maximum fitness values for different network sizes.

4.2 Optimizing for performance and reconfiguration costs

This experiment compares the relative quality of solutions evolved by Hermes with those obtained by component-dependency analysis when the main objective is to minimize reconfiguration costs while maximizing reconfiguration performance, i.e., $\alpha_{cost} = 0.4$, $\alpha_{perf} = 0.4$, and $\alpha_{rel} = 0.2$. Such trade-off preferences may arise in scenarios where the reconfiguration is driven by variations in system performance rather than by failures that may threaten the functionality of the system. For example, communication paths between remote data mirrors may be reconfigured at run time as environmental conditions such as throughput and loss rate change. For all following experiments, the starting network of remote data mirrors comprises 25 components and at least 35 communication links.

Figure 7 plots the average maximum fitness values of adaptation paths evolved by Hermes per generation. Solutions evolved by Hermes achieve an approximate fitness value of 3398. In contrast, adaptation paths generated by component-dependency analysis achieve an approximate fitness value of 1536, which is represented by the filled circle plotted at generation 0 before Hermes modifies it. In general, Hermes evolved solutions that maximized performance without significantly increasing reconfiguration costs and thus improved fitness by 220%. To achieve this objective, Hermes reordered sets of reconfiguration instructions to sequentially reconfigure small subsets of remote data mirrors and connections at any given time, thereby enabling the majority of remote data mirrors to continue propagating data in the meantime. On the average, Hermes increased reconfiguration costs by approximately 12 seconds, less than a 3% increase. As such, the 220% difference in fitness values emphasizes how reordering the initial adaptation path may improve a reconfiguration's performance by reducing system disruption during reconfiguration. Lastly, as this plot illus-

trates, Hermes achieved large fitness gains within the first 500 generations (< 30 seconds), suggesting that tradeoffs may be balanced in a reasonable amount of time within the context of remote data mirroring.

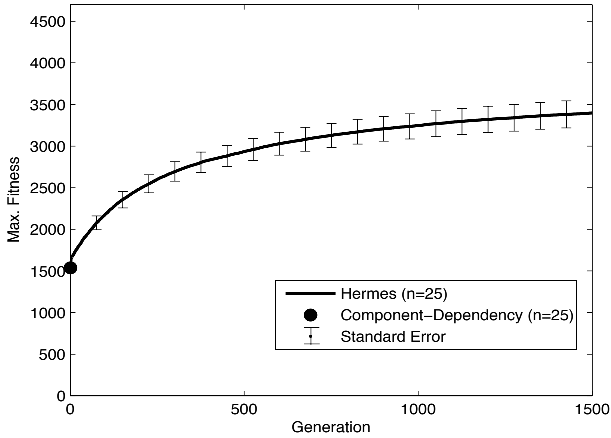

Figure 7: Progression of average fitness values when minimizing reconfiguration costs and maximizing reconfiguration performance.

Finally, Figure 8 plots the average amount of data (in MB) sent and queued by remote data mirrors during reconfiguration. This plot is generated by analyzing evolved adaptation paths to determine the time period in which a remote data mirror is either in active or passive mode. As this plot illustrates, Hermes gradually evolves solutions that diffuse larger amounts of data, while queueing less data. These results confirm that adaptation paths are multi-dimensional. Furthermore, these results also suggest the inherent tradeoff between the performance and reliability of a reconfiguration. Specifically, minimizing system disruption enables remote data mirrors to diffuse greater amounts of data, but a single failure during reconfiguration could potentially lose significant amounts of data.

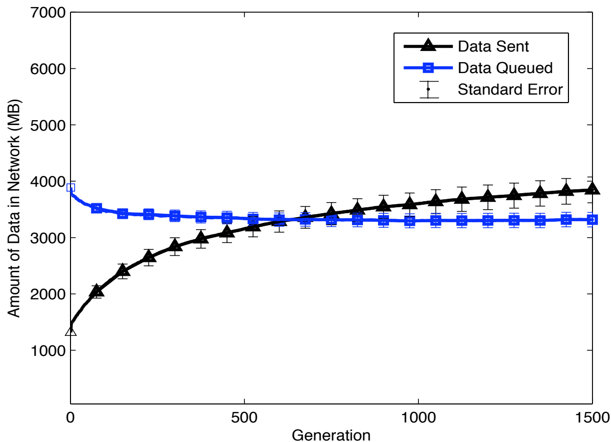

Figure 8: Performance and reliability tradeoffs in evolved solutions.

4.3 Optimizing for Reliability

This experiment compares the relative quality of solutions evolved by Hermes with those obtained by component-dependency analysis when the main objective is to maximize the reliability of a reconfiguration, i.e., $\alpha_{\text{cost}} = 0.2$, $\alpha_{\text{perf}} = 0.2$, and $\alpha_{\text{rel}} = 0.6$. Such trade-off preferences may arise when the reconfiguration is driven by failures that threaten the functionality of the system rather than by variations in system performance. For example, the failure of either a remote data mirror or a connection between remote data mirrors may cause data to be permanently lost. As such, this experiment explores scenarios where the cost of losing data is severe.

Figure 9 plots the average maximum fitness values of adaptation paths per generation. Solutions evolved by Hermes achieve an approximate fitness value of 4502, a 139% improvement over component-dependency analysis whose fitness is represented by the filled circle plotted at generation 0, before Hermes modifies the initial adaptation path. Hermes achieves higher fitness values by evolving solutions different from the initial adaptation path in two key ways. First, Hermes adds pairs of "*passivate*" and "*activate*" instructions not present in the initial adaptation path. Second, Hermes reorders the sequence of reconfiguration instructions to establish large regions of quiescence throughout most of the reconfiguration. Specifically, passivate instructions are shifted to the beginning of the adaptation path, thereby temporarily pausing most remote data mirrors. Hermes then reconfigures the network of remote data mirrors before finally setting remote data mirrors back to active mode. Although this strategy is similar to Kramer and Magee's dynamic change management protocol, Hermes also makes tradeoffs with factors such as performance and cost. Lastly, this plot also illustrates how Hermes achieves large fitness gains within the first 500 generations (< 30 seconds), which further reaffirm our findings that Hermes can balance competing tradeoffs in a reasonable amount of time within the context of remote data mirrors.

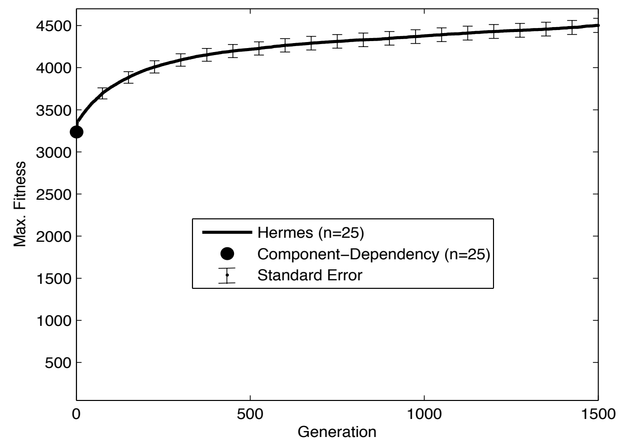

Figure 9: Progression of average fitness values when maximizing reconfiguration reliability.

In contrast to the previous experiment, where minimizing reconfiguration disruption was one of the primary concerns, in this experiment the cost of reconfiguration increased in order to maximize the reliability of the reconfiguration. As

Figure 10 illustrates, Hermes increased reconfiguration costs, on the average, by approximate 33 seconds more than the initial adaptation path, an increase of approximate 12%. This increase in reconfiguration cost is a direct result of the additional passivate and activate instructions inserted by Hermes. Interestingly, these paris of additional instructions were sometimes applied to remote data mirrors not involved in the reconfiguration process. Were it not for these additional instructions, these remote data mirrors would have propagated data during the entire reconfiguration. Thus, by passivating most remote data mirrors, Hermes provided better data reliability at the expense of higher reconfiguration costs.

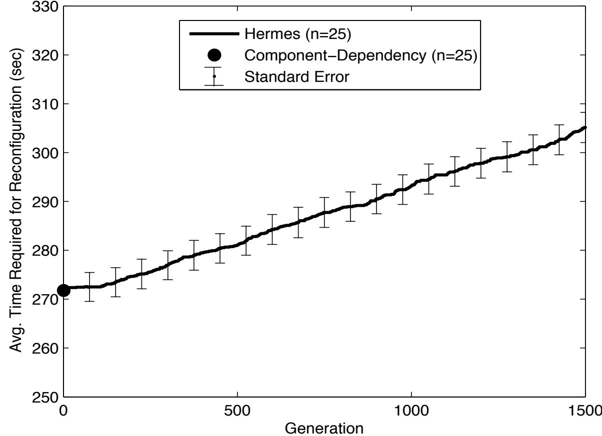

Figure 10: Average time required to complete reconfiguration when maximizing reconfiguration reliability.

Figure 11 plots the amount of data (in MB) sent and queued by remote data mirrors during reconfiguration. Data sent measures the amount of data remote data mirrors diffused during reconfiguration as a result of being *active*. Likewise, data queued measures the amount of data produced but not diffused by remote data mirrors because their operational status was set to *passive* state. As this plot illustrates, Hermes gradually evolves solutions that queue larger amounts of data than is diffused throughout the network. Although this plot suggests greater amounts of data production and diffusion during reconfiguration when compared to Figure 8, this difference is a result of longer times required for the reconfiguration to complete. The longer the reconfiguration, the greater the amount of data produced and diffused, even though fewer remote data mirrors are in active mode. Lastly, these results also confirm the tradeoffs observed in the previous experiments between the performance and reliability of adaptation paths. Namely, maximizing reliability typically implies a higher level of system disruption. While it is generally undesirable to disrupt system services by passivating large numbers of remote data mirrors, doing so creates a region of quiescence that better protects data against failures during reconfiguration.

5. DISCUSSION

The experiments presented in this paper confirm that adaptation paths are multi-dimensional and that the specific sequence of reconfiguration instructions produce non-linear

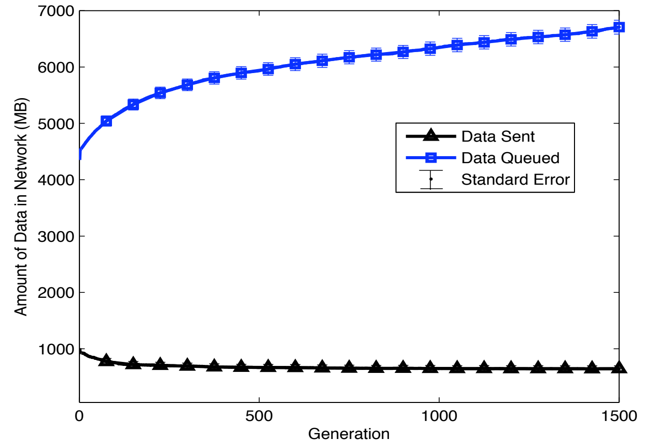

Figure 11: Reconfiguration and performance tradeoffs in evolved solutions.

effects upon the cost, performance, and reliability of a reconfiguration. In each experiment trial, adaptation paths evolved by Hermes achieved a higher fitness value than those produced by component-dependency analysis, with a significance of $p < 0.01$. As a result, we reject our null hypothesis, H_0. Furthermore, we accept our alternate hypothesis, H_1 and conclude that Hermes is capable of evolving higher quality adaptation paths when compared to those generated by component-dependency analysis while also balancing multi-dimensional tradeoffs between non-functional requirements.

In terms of execution times, Hermes terminated within 2 minutes or less. In this amount of time, Hermes evaluated approximately 1.5 million candidate adaptation paths. Furthermore, Hermes typically achieved large gains in fitness within the first 500 generations (30 seconds). In this relatively short amount of time, Hermes is able to automatically find safe adaptation paths that balance competing objectives better than what other techniques currently produce. Moreover, since Hermes always starts with a viable safe adaptation path generated by component-dependency analysis, the solutions produced by Hermes can only improve. As a result, Hermes does not require a predetermined amount of execution time before a viable safe adaptation path is available.

6. CONCLUSIONS

In this paper we presented Hermes, an approach for evolving safe adaptation paths while balancing competing objectives in non-functional requirements. By rescaling the relative importance of each concern, Hermes can evolve different types of solutions in response to changing requirements and environmental conditions. While Hermes can be applied at design time to explore larger sets of safe adaptation paths, it may also be applied at run-time. Future work includes extending Hermes to exploit potential parallelism between reconfiguration instructions, as well as exploring how different reconfiguration instruction costs affect the quality of solutions evolved by Hermes. Lastly, we plan on extending and applying Hermes to different application domains, including different architectures such as multi-tiered and virtual machine infrastructures.

7. ACKNOWLEDGEMENTS

This work has been supported in part by NSF grants CCF-0541131, IIP-0700329, CCF-0750787, CCF-0820220, CNS-0854931, CNS-0751155, CNS-0915855, Army Research Office grant W911NF-08-1-0495, Ford Motor Company, and a Quality Fund Program grant from Michigan State University.

8. REFERENCES

[1] M. Brameier and W. Banzhaf. *Linear Genetic Programming*. Number XVI in Genetic and Evolutionary Computation. Springer, 2007.

[2] S. W. Cheng, D. Garlan, and B. Schmerl. Architecture-based self-adaptation in the presence of multiple objectives. In *Proceedings of the 2006 International Workshop on Self-adaptation and Self-Managing Systems*, pages 2–8, Shanghai, China, 2006. ACM.

[3] D. Garlan, S. W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, 2004.

[4] D. Garlan, B. Schmerl, and J. Chang. Using gauges for architecture-based monitoring and adaptation. In *In the Proceedings of the Working Conference on Complex and Dynamic Systems Architecture*, Brisbane, Australia, December 2001 2001.

[5] H. J. Goldsby and Betty H.C. Cheng. Automatically generating behavioral models of adaptive systems to address uncertainty. In *Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems*, pages 568–583 (Distinguised Paper Award), Berlin, Heidelberg, 2008. Springer-Verlag.

[6] H. J. Goldsby, Betty H.C. Cheng, P. K. McKinley, D. B. Knoester, and C. A. Ofria. Digital evolution of behavioral models for autonomic systems. In *Proceedings of the Fifth IEEE International Conference on Autonomic Computing*, pages 87–96 (Best Paper Award), Chicago, Illinois, 2008. IEEE Computer Society.

[7] J. H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, MA, USA, 1992.

[8] M. Ji, A. Veitch, and J. Wilkes. Seneca: Remote mirroring done write. In *USENIX 2003 Annual Technical Conference*, pages 253–268, Berkeley, CA, USA, June 2003. USENIX Association.

[9] G. Kaiser, P. Gross, G. Kc, and J. Parekh. An approach to autonomizing legacy systems. In *Proceedings of the First Workshop on Self-Healing, Adaptive, and Self-MANaged Systems*, 2002.

[10] E. P. Kasten and P. K. McKinley. MESO: Supporting online decision making in autonomic computing systems. *IEEE Transactions on Knowledge and Data Engineering*, 19(4):485–499, April 2007.

[11] K. Keeton, C. Santos, D. Beyer, J. Chase, and J. Wilkes. Designing for disasters. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, pages 59–62, Berkeley, CA, USA, 2004. USENIX Association.

[12] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. The MIT Press, December 1992.

[13] J. Kramer and J. Magee. The evolving philosophers problem: Dynamic change management. *IEEE Trans. on Soft. Eng.*, 16(11):1293–1306, 1990.

[14] J. Kramer and J. Magee. Self-managed systems: an architectural challenge. In *Future of Software Engineering 2007*, pages 259–268, Minneapolis, Minnesota, May 2007. IEEE Computer Society.

[15] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and Betty H.C. Cheng. Composing adaptive software. *Computer*, 37(7):56–64, 2004.

[16] M. Mikalsen, N. Paspallis, J. Floch, E. Stav, G. A. Papadopoulos, and A. Chimaris. Distributed context management in a mobility and adaptation enabling middleware. In *SAC'06: Proc. of the 2006 ACM symposium on Applied Computing*, pages 733–734, New York, NY, USA, 2006. ACM.

[17] H. Newman, I. Legrand, P. Galvez, R. Voicu, and C. Cistoiu. MonALISA: A Distributed Monitoring Service Architecture. In *Proceedings of the 2003 Conference for Computing in High Energy and Nuclear Physics*, March 2003.

[18] A. J. Ramirez and Betty H.C. Cheng. Evolving models at run time to address functional and non-functional adaptation requirements. In *Proceedings of the Fourth Workshop on Models at Run Time*, volume 509, pages 31–40, Denver, Colorado, USA, October 2009. ACM.

[19] A. J. Ramirez, D. B. Knoester, Betty H.C. Cheng, and P. K. McKinley. Applying genetic algorithms to decision making in autonomic computing systems. In *Proceedings of the Sixth International Conference on Autonomic Computing*, pages 97–106 (Best Paper Award), Barcelona, Spain, June 2009.

[20] S. M. Sadjadi and P. K. McKinley. ACT: An adaptive CORBA template to support unanticipated adaptation. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, pages 74–83, 2004.

[21] Y. Vandewoude and P. Ebraert. Tranquillity: A low disruptive alternative to quiescence for ensuring safe dynamic updates. *IEEE Transactions on Software Engineering*, 33(12):856–868, December 2007.

[22] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das. Utility functions in autonomic systems. In *Proceedings of the First IEEE International Conference on Autonomic Computing*, pages 70–77, New York, NY, USA, 2004. IEEE Computer Society.

[23] J. Zhang and Betty H.C. Cheng. Model-based development of dynamically adaptive software. In *Proceedings of the 28th International Conference on Software Engineering*, pages 371–380, New York, NY, USA, 2006. ACM (Distinguished Paper Award).

[24] J. Zhang, Betty H.C. Cheng, Z. Yang, and P. K. McKinley. *Enabling safe dynamic component-based software adaptation*, volume 3549 of *Lecture Notes in Computer Science*, pages 194–211. Springer Berlin / Heidelberg, 2005.