Evolving Models at Run Time to Address Functional and Non-Functional Adaptation Requirements*

Andres J. Ramirez and Betty H.C. Cheng

Michigan State University
Department of Computer Science and Engineering,
3115 Engineering Building, East Lansing, MI 48824
{ramir105,chengb}@cse.msu.edu

Abstract. Increasingly, applications need to dynamically self-reconfigure as new environmental conditions arise at run time. In order to self-reconfigure, an adaptive system must determine which target system configuration will yield the desired behavior based on current execution conditions. However, it may be impractical to evaluate all potential system configurations in a reasonable time frame. This paper presents a model-based approach that leverages evolutionary computation to automatically generate, at run time, target system models that balance tradeoffs between functional and non-functional requirements in response to run-time monitoring of environmental conditions. Specifically, this approach generates graph-based representations of architectural models for potential target system configurations. The current run-time system models serve to constrain the degree of change and novelty in the newly generated models. This approach is applied to the dynamic reconfiguration of a set of remote data mirrors, where operational and reconfiguration costs are minimized, while maximizing data reliability and network performance.

Key words: dynamic reconfiguration, non-functional requirements, evolutionary computation, run-time models.

1 Introduction

It is increasingly important for applications to dynamically adapt as requirements change and new environmental conditions arise [1]. In addition, it is important for adaptive systems to self-reconfigure with little or no human input to help prevent costly downtimes while code is being modified. To address this concern, IBM proposed autonomic computing where a system manages itself to achieve a system administrator's high-level goals through self-* properties such as self-configuration and self-optimization [2]. To self-reconfigure, an adaptive system must automatically determine which target system configuration will

* This work has been supported in part by NSF grants CCF-0541131, CNS-0551622, CCF-0750787, CNS-0751155, IIP-0700329, and CCF-0820220, Army Research Office W911NF-08-1-0495, Ford Motor Company, and a grant from Michigan State University's Quality Fund.

yield the desired behavior in response to current system and environmental conditions, while also taking into consideration tradeoffs between functional and non-functional requirements. It may be impractical, however, to evaluate all potential target systems in a reasonable amount of time. This paper presents a model-based approach that leverages evolutionary computation to generate, at run time, target system models that balance tradeoffs between functional and non-functional requirements in response to changing environmental conditions.

Self-adaptive systems comprise three key enabling technologies: monitoring, decision-making, and reconfiguration. Monitoring enables an application to be aware of its environment to detect conditions that warrant reconfiguration. Decision-making analyzes monitoring information to determine how the application should be reconfigured. Reconfiguration enables an application to modify itself to fulfill its requirements. Many self-adaptive systems apply model-based techniques to determine which target system configuration will yield the desired system behavior in response to current environmental conditions [3–7]. While powerful, these approaches typically use scenarios identified at *design time* to guide self-reconfigurations. Furthermore, as the complexity of adaptive logic grows, maintaining the set of models and reconfiguration plans may become unmanageable and potentially inconsistent. Recently, researchers have applied evolutionary computation techniques to the design of self-adaptive systems [8–10]. While these approaches enable developers to explore richer sets of behavioral models that satisfy system requirements, they are only applicable at design time due to the significant amount of time required to generate these models.

This paper presents Plato-MDE, an evolutionary computation-based approach for generating target system models at *run-time* in response to changing requirements and environmental conditions. Each target system model represents a potential system configuration that may be reached through a sequence of reconfiguration steps. Plato-MDE evaluates each generated target system model to determine its suitability given current system conditions. In addition, Plato-MDE leverages current system models to constrain the degree of change in the generated target models. As a result, Plato-MDE enables an adaptive system to implicitly control the complexity and novelty of the reconfiguration itself at run time. Moreover, rather than prescribing explicit reconfiguration plans at design time in anticipation of possible reconfiguration scenarios, developers need only specify the relative importance of each functional and non-functional concern to apply Plato-MDE.

Plato-MDE supports a model-based approach that leverages information from run-time system models to optimize the generation of target system models. In particular, Plato-MDE applies domain-independent evaluation functions to compare the structure and configuration of each generated target system model against a current architectural model of the executing system. The structural and configuration differences identified from this analysis enable Plato-MDE to implicitly constrain the novelty of generated target system models, and thus control the complexity and cost of the reconfiguration itself. For example, to minimize reconfiguration costs, Plato-MDE might focus on generating target system mod-

els that are structurally similar to the current system model, thereby reducing the number of structural changes required to reconfigure the system. Plato-MDE accomplishes these objectives at run-time by applying *genetic algorithms* [11] to automatically balance tradeoffs in functional and non-functioanl requirements. As a result, Plato-MDE evolves target system models at run-time, where better solutions tend to eventually dominate the solution space.

We applied Plato-MDE to the dynamic reconfiguration of an overlay network for diffusing data to a collection of remote data mirrors [12]. Specifically, Plato-MDE was able to evolve target system models that not only maintained connectivity across the network of remote data mirrors such that data could be diffused to every node, but also minimized operational and reconfiguration costs while maximizing data reliability and network performance. Furthermore, Plato-MDE was able to leverage run-time system models to control the complexity and novelty of the generated target system reconfigurations, thus implicitly controlling reconfiguration costs at run time. The remainder of this paper is organized as follows. Section 2 overviews genetic algorithms. Section 3 overviews Plato-MDE. In Section 4 we present a case study in which we apply Plato-MDE and provide preliminary results. Section 5 compares Plato-MDE to other self-adaptation approaches. Lastly, Section 6 summarizes our results and presents future work.

2 Background: Genetic Algorithms

A genetic algorithm is a stochastic search-based technique for optimization problems that comprises a population of individuals, each encoding a candidate solution in a chromosome representation [11]. Fitness functions are used in each iteration of the algorithm to evaluate an individual's encoded solution. This fitness information enables a genetic algorithm to select a subset of promising individuals for further processing. Specifically, the operation of *crossover* is used to exchange building blocks between two fit individuals, hopefully creating offspring with higher fitness values than either parent. The crossover operator loses genetic variation in a population throughout generations, possibly leading to premature convergence and suboptimal solutions. In order to counter this effect, *mutation* re-introduces genetic variation by randomly changing parts of an individual's encoded solution according to specific mutation rates [11]. Generally, genetic algorithms are executed until the algorithm converges upon a single solution or the alloted execution time is exceeded.

3 Plato-MDE

Plato-MDE is a genetic algorithm-based approach developed for generating target system models at run time in response to changing environmental conditions, while balancing tradeoffs between functional and non-functional requirements. Plato-MDE extends Plato [13] with a model-based approach that focuses on generating architectural models and properties of the connectors between the components at run time. In contrast, Plato did not consider structural differences between the current application's architecture and the generated target system

reconfiguration models. This extension enables Plato-MDE to implicitly control the cost of a reconfiguration at run time. As the data flow diagram (DFD) in Figure 1 illustrates, several inputs and configurations must be supplied in order to apply Plato-MDE to the decision-making process of a self-adaptive system. At a high-level of abstraction, Plato-MDE accepts data from the monitoring infrastructure and outputs a set of target system models that specify new suitable reconfigurations. As an initialization step, developers must first configure Plato-MDE for the application's domain and specify how the quality of a target system model should be evaluated. Next, we describe the use of Plato-MDE in detail.

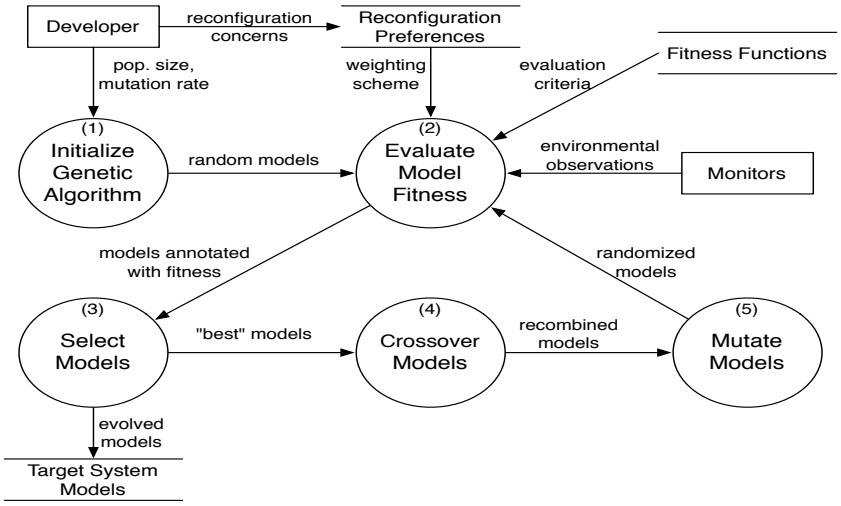

Fig. 1. Data Flow Diagram for Plato-MDE

Step 1. Initialize Genetic Algorithm. To apply Plato-MDE, a developer must configure operational parameters that determine how the genetic algorithm will execute as well as what will be used to assess the quality of the output it produces. In particular, developers must specify parameters such as population size, maximum number of generations, and mutation rates for Plato-MDE. The population size reflects how many potentially different target system models are being examined at any single point in time. Similarly, the number of generations limit the amount of time that Plato-MDE may spend generating target system models. Lastly, mutation rates indicate the degree of randomness that Plato-MDE will apply to generate target system models. Experimentation is typically required to discover suitable parameters for different application domains.

The first step in Plato-MDE, highlighted in Figure 1 as *(1)*, creates a population of random individuals. In Plato-MDE, each individual encodes a graph-based model representation of the target application's architecture. Specifically, the application is abstracted to a set of components and their interconnections, both annotated with sets of reconfigurable properties that describe their configurations and state. For example, a property in a networked application may specify whether a link is active or not and which communication protocol is currently

selected. This representation, similar to architectural models, is appropriate for abstracting relevant details of the executing system [3, 5, 7].

Step 2. Evaluate Model Fitness. Fitness functions, akin to utility functions, are used to map an individual's encoded solution to a numerical value proportional to its overall quality [11]. As Figure 1 illustrates in *(2)*, Plato-MDE applies fitness functions to assess the suitability of a particular target system model based on current system conditions supplied by the application's monitoring infrastructure. Plato-MDE applies domain-dependent fitness functions to evaluate target system models from a domain-specific perspective, such as approximating the performance and reliability of a specific network based on a protocol's configuration. In addition, Plato-MDE applies domain-independent fitness functions to evaluate target system models from structural and behavioral perspectives. For instance, Plato-MDE can approximate reconfiguration costs by identifying the structural and configurational changes between the current system model and the generated target system reconfiguration. Therefore, to minimize reconfiguration costs at run time, Plato-MDE could assign higher fitness values to target system models whose structure and configuration are most similar to the current system model.

Developers may also supply a weighting scheme that will be associated with specific fitness functions to indicate the relative importance of different reconfiguration priorities. Moreover, developers can also introduce high-level code to rescale the weighting scheme of individual fitness functions if requirements are likely to change while the application executes. Updating reconfiguration priorities at run time enables Plato-MDE to generate different reconfiguration plans that address changes in requirements.

Step 3. Selection. A selection strategy determines which individuals in the population should be explored further in future generations. As step *(3)* in Figure 1 illustrates, Plato-MDE applies a tournament selection strategy [11] to determine which target system models to compare. Specifically, two target system models are selected at random from the population and their relative fitness value is compared. Whichever target system model has a higher fitness value *survives* and moves onto the next generation. This selective pressure, similar to natural selection in living organisms, drives Plato-MDE to concentrate its search towards more promising target system models that are suitable for current system conditions. Once the maximum number of generations are executed, the most fit target system model is selected as the result.

Step 4. Crossover. The goal of the crossover operator is to construct new solutions by recombining key building blocks from existing solutions in the current population [11]. Similarly, as Figure 2 illustrates, Plato-MDE applies a customized crossover operator that works on architectural models by exchanging the key elements between two target system models, referred to as parents, to produce two potentially new offspring target system models at run time. Specifically, the Plato-MDE crossover operator generates two new target system models by randomly exchanging the components, interconnections, and properties of both parents and recombining them into offspring individuals. As a result, the

crossover operator enables Plato-MDE to combine elements of good solutions to form even better solutions.

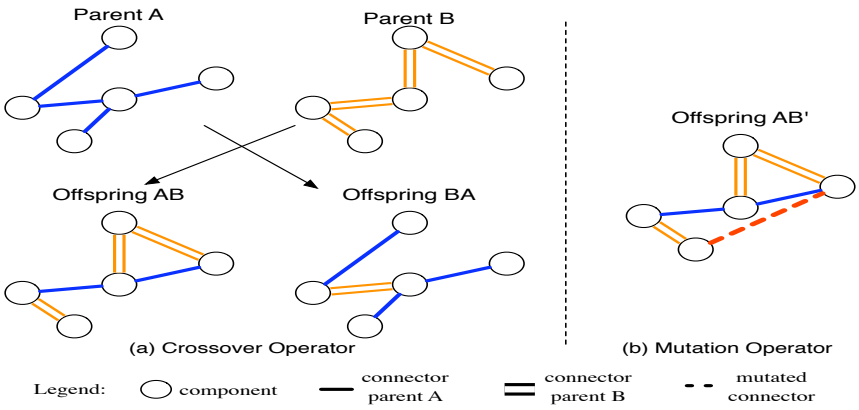

Fig. 2. Crossover and Mutation operators in Plato-MDE

Step 5. Mutation. The goal of the mutation operator is to introduce variation into the population and prevent premature convergence [11]. As Figure 1 shows in step *(5)*, Plato-MDE applies a custom mutation operator to randomly change properties of components and interconnections in an architectural model. Specifically, the mutation operator accepts an architectural model as input and randomly reassigns component and interconnection properties. For example, Figure 2(b) shows how a previously nonexistent interconnection has been created between two components (dashed line) in the architectural model. As a result, the mutation operator enables Plato-MDE to explore additional target system models at run time that cannot be generated solely through the crossover operator.

4 Case Study

This section presents a case study where we use Plato-MDE within a simulated industrial application whose primary objective is to diffuse data to a set of 25 remote data mirrors [12] across dynamic and unreliable networks. In this application, Plato-MDE generates target system models of an overlay network used to diffuse data to every remote data mirror. In contrast to previous experiments [13], this case study leverages run-time system models to constrain the degree of change involved in a particular reconfiguration. Note that the experiment presented in this section was executed on a MacBook Pro with a 2.53GHz Intel Core 2 Duo Processor and 4GB of RAM. In addition, we performed 30 trials of the experiment, for statistical purposes, and present the averaged results.

4.1 Remote Data Mirroring

Remote data mirroring is a technique for duplicating and storing data at one or more secondary sites to physically isolate copies from failures that may affect the primary copy [12]. A key benefit of remote data mirroring is that important

data continues to be accessible even if one copy is lost or becomes unreachable. Designing and deploying remote data mirror solutions, however, is a complex task due to the competing objectives of maximizing performance while minimizing operational costs and data loss potential [12]. For instance, each network link used to propagate data incurs an operational cost and is characterized by measurable throughput, latency, and loss rates. Moreover, each network link distributes data in one of two propagation modes. In *synchronous* propagation the secondary site receives and applies each write before the write completes at the primary site [12]. In *asynchronous* propagation, updates are batched and periodically distributed to secondary sites. While synchronous propagation provides better data reliability than asynchronous propagation, it tends to consume large amounts of network bandwidth in the process. In contrast, asynchronous propagation fails to provide the same level of reliability as synchronous propagation, but tends to achieve better network performance.

In this case study we apply Plato-MDE to dynamically reconfigure a set of 25 remote data mirrors diffusing data across a dynamic and unreliable network. In particular, Plato-MDE must maintain connectivity across the network of remote data mirrors while minimizing operational and reconfiguration costs, and maximizing data reliability and network performance.

4.2 Applying Plato-MDE to Remote Data Mirroring

In Plato-MDE, every individual in the population encodes an architectural model that specifies potential reconfigured target systems. For this case study, each component in the encoded architectural models represents a remote data mirror capable of producing data at a specific rate. Similarly, every interconnection in the encoded architectural models represents an overlay network link capable of propagating data between remote data mirrors. Therefore, in addition to specifying whether each connection is active or inactive, each connection is also associated with one of seven possible propagation methods [13]. It is important to note that with n overlay network links and m propagation methods, over $2^{\frac{n(n-1)}{2}} * m^n$ potential configurations exist. Thus, with a complete overlay network of 25 remote data mirrors, approximately $7^{300} * 2^{300}$ potential target system models exist, far too many configurations to exhaustively evaluate in a reasonable amount of time.

Plato-MDE extracts data from the application's monitoring infrastructure to maintain an architectural model of the executing system. Many different metrics can be gathered, however, for this case study, the monitoring infrastructure measures the throughput, latency, bandwidth, and data loss rates of each overlay network link that can be used to propagate data between remote data mirrors. Plato-MDE leverages this current system model to evaluate each generated target system model and approximate the effects of different network configurations. To this end, we applied a set of domain-dependent fitness functions to evaluate network configurations in terms of operational costs, network performance, and data reliability. Plato-MDE also applied simple model checks to ensure generated system models did not violate either budget or connectivity constraints. In addition, domain-independent fitness functions compute the degree of change

between pairs of architectural models by identifying the structural and configurational changes between them, enabling Plato-MDE to implicitly control reconfiguration costs.

4.3 Experimental Results

The goal for the initial overlay network design was to minimize operational costs, possibly at the expense of incurring poor network performance and data reliability. To generate this type of network, we supplied Plato-MDE with a vector of reconfiguration priorities where all coefficients were set to zero except for cost. As Figure 3(a) illustrates, Plato-MDE produced a spanning tree overlay network where every node is connected but no link redundancy is provided. This overlay network design minimizes operational costs by activating the minimum number of network links required to maintain connectivity and enable remote data mirrors to diffuse data. However, this overlay network design does not provide much data reliability. In particular, a single link failure in the overlay network would disconnect the set of remote data mirrors and data may be lost.

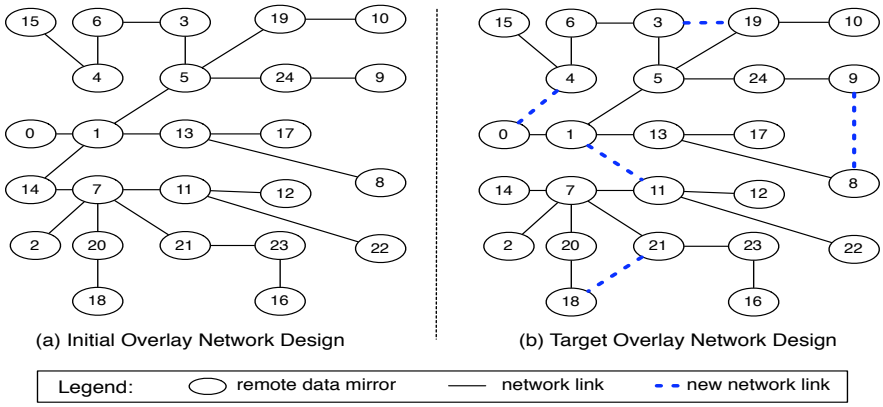

Fig. 3. Source and Target Network Design Models.

Next, we randomly selected an active link propagating data in the initial overlay network and set its operational status to *faulty*. This network link state corresponds to a link failure, thereby disconnecting the network of remote data mirrors and prompting Plato-MDE to reconfigure the overlay network. While Plato-MDE could have been invoked with the same vector of reconfiguration priorities to generate another overlay network design that re-establishes connectivity across the set of remote data mirrors while minimizing operational costs, the reconfiguration priorities were rescaled in an attempt to prevent future link failures from disconnecting the set of remote data mirrors. Specifically, the new vector of reconfiguration priorities changed the importance of minimizing operational costs to 12%, maximizing network performance to 12%, maximizing data reliability to 38%, and target model similarity to 38%. With this new vector of reconfiguration priorities, Plato-MDE produced target system models that reused the underlying network structure while adding redundant links and setting most propagation methods to synchronous mode.

Figure 3(b) shows an example overlay network design produced by Plato-MDE to re-establish connectivity within the set of remote data mirrors. This target system model satisfies the two primary design concerns specified in the vector of reconfiguration priorities: increased data reliability and reduced reconfiguration overhead. By increasing the importance of data reliability, Plato-MDE generated overlay networks with redundant links and set most propagation methods in the overlay network links to synchronous mode. Figure 3 also illustrates how the target overlay network reuses a significant portion of the underlying initial network. While Plato [13] would generate target system models without taking into account the complexity or cost of the reconfiguration, Plato-MDE preserved most of the initial network's structure to implicitly reduce the cost of reconfiguration at run time. Plato-MDE took approximately 30 seconds or less to begin converging upon suitable target system reconfigurations, which is within the acceptable time frame for remote data mirroring.

5 Related Work

Several approaches for enabling self-adaptive behavior leverage architectural models at run time to evaluate system conditions and select the most suitable reconfiguration in response to current environmental conditions. For instance, the C2 framework [7] applies software architectural models to plan, coordinate, and implement reconfigurations at run time. In addition, both the Performance Management Framework (PMF) [5] and the Rainbow Adaptation Framework [3, 4] instantiate architectural models with run-time monitoring information to determine when and how to reconfigure a system. While Plato-MDE adopts a similar approach for determining how the application should be reconfigured, several key differences exist. For instance, Plato-MDE is capable of generating *any* target system model reachable through a series of reconfiguration steps. In contrast, C2 [7] relies on a repository of pre-generated target models, and PMF [5] and Rainbow [3, 4] generate new target models through predetermined combinations of their reconfiguration steps. Furthermore, C2 [7], PMF [5], and Rainbow [3, 4] encode their reconfiguration priorities at design time. Plato-MDE, on the other hand, can update reconfiguration preferences at run time to address changes in requirements and environmental conditions. Lastly, while PMF [5] and Rainbow [3, 4] evaluate the utility of target reconfigurations to predict their impact upon the system, Plato-MDE also leverages this utility information to guide the search towards more promising target system models.

6 Conclusions

We have presented Plato-MDE, a model-based approach that leverages evolutionary computation to generate, at run time, target system models that balance tradeoffs between functional and non-functional requirements in response to current system conditions. Plato-MDE extends Plato [13] with domain-independent model-based fitness functions that analyze the structural differences between current and target system models to implicitly control reconfiguration costs at run time. We have successfully applied Plato-MDE to the dynamic reconfiguration of a set of remote data mirrors, where generated target system models

enable data diffusion among remote data mirrors by maintaining network connectivity while minimizing costs and maximizing network performance and data reliability. Future directions for this work include exploring how to decentralize the architecture of Plato-MDE to reduce potential performance bottlenecks.

References

1. McKinley, P.K., Sadjadi, S.M., Kasten, E.P., Betty H.C. Cheng: Composing adaptive software. Computer **37**(7) (2004) 56–64
2. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**(1) (2003) 41–50
3. Garlan, D., Cheng, S.W., Huang, A.C., Schmerl, B., Steenkiste, P.: Rainbow: Architecture-based self-adaptation with reusable infrastructure. Computer **37**(10) (2004) 46–54
4. Cheng, S.W., Garlan, D., Schmerl, B.: Architecture-based self-adaptation in the presence of multiple objectives. In: Proceedings of the 2006 International Workshop on Self-adaptation and Self-Managing Systems, New York, NY, USA, ACM (2006) 2–8
5. Caporuscio, M., Marco, A.D., Inverardi, P.: Model-based system reconfiguration for dynamic performance management. Journal of Systems and Software **80**(4) (September 2007) 455–473
6. Mikalsen, M., Paspallis, N., Floch, J., Stav, E., Papadopoulos, G.A., Chimaris, A.: Distributed context management in a mobility and adaptation enabling middleware (madam). In: SAC'06: Proc. of the 2006 ACM symposium on Applied Computing, New York, NY, USA, ACM (2006) 733–734
7. Oreizy, P., Gorlick, M., Taylor, R.N., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D.S., Wolf, A.L.: An Architecture-Based Approach to Self-Adaptive Software. IEEE Intelligent Systems **14**(3) (1999) 54–62
8. Goldsby, H.J., Betty H.C. Cheng, McKinley, P.K., Knoester, D.B., Ofria, C.A.: Digital evolution of behavioral models for autonomic systems. In: Proceedings of the fifth IEEE International Conference on Autonomic Computing, Washington, DC, USA, IEEE Computer Society (2008) 87–96 (Best Paper Award)
9. Goldsby, H.J., 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, Berlin, Heidelberg, Springer-Verlag (2008) 568–583 (Distinguised Paper Award)
10. Knoester, D.B., Ramirez, A.J., Cheng, B. H.C., McKinley, P.K.: Evolution of robust data distribution among digital organisms. In: Proceedings of the 11th annual conference on Genetic and Evolutionary Computation (GECCO '09), Montreal, Canada (July 2009) 137–144 (Nominated for Best Paper)
11. Holland, J.H.: Adaptation in Natural and Artificial Systems. MIT Press, Cambridge, MA, USA (1992)
12. Ji, M., Veitch, A., Wilkes, J.: Seneca: Remote mirroring done write. In: USENIX 2003 Annual Technical Conference, Berkeley, CA, USA, USENIX Association (June 2003) 253–268
13. Ramirez, A.J., Knoester, D.B., Cheng, B. H.C., McKinley, P.K.: Applying genetic algorithms to decision making in autonomic computing systems. In: Proceedings of the Sixth International Conference on Autonomic Computing (ICAC'09), Barcelona, Spain (June 2009) 97–106 (Best Paper Award)