Proceedings of the 3rd International Conference on Systems and Control, Algiers, Algeria, October 29-31, 2013

A new evolutionary approach to decision-making in autonomic systems

Abdelghani Alidra, Mohamed Tahar Kimour

Abstract— **Increasingly, autonomic systems are present in our lives. For this kind of systems the ability to self-reconfigure and adapt in response to changes in users requirements and environmental conditions is primordial. Several approaches have been proposed in the literature to achieve self-reconfiguration, however, as the complexity of the adaptive system grows, designing and managing the set of reconfiguration rules becomes difficult and error-prone. To tackle this limitation, we propose a new approach that uses a search-based evolutionary algorithm that explores valid configurations to find the most relevant one given a specific running context. Another salient advantage of our approach is the re-exploitation, in the context of adaptability, of the design knowledge and existing model-based technologies through the reuse of the feature model of the system.**

I. Introduction

Autonomic computing is a paradigm that has been proposed in response to the growing complexity of today's software systems. Indeed, software systems are becoming increasingly distributed, open and connected and it is more and more difficult to anticipate the system's requirements regarding the user needs and the changing environmental conditions. Autonomic computing refers to "any system that manages itself based on a system administrator's high-level objectives while incorporating capabilities such as self-reconfiguration and self-optimization" [1].

Traditionally, reconfiguration rules were explicitly written at design time taking in account the environmental running conditions and matching particular events to specific reconfiguration actions [2,3,4,5]. However, proceeding this way enables the adaptation only against scenarios that were considered at design time. Besides, as the complexity of the adaptive logic grows, designing, managing and maintaining the set of reconfiguration rules can quickly become problematic. To overcome these limitations, we propose an approach that implements the decision-making process using a genetic algorithm. The genetic algorithm calculates the (near) optimal system's configuration given a set of well expressed objective functions according to a set of environmental variables instead of writing reconfigurations rules. To this end, we reuse the feature model to guide the expression of the decision-making problem and to constraint the validity of new found configurations.

Abdelghani Alidra is with the computer science department, 20th August 1955 University - Skikda, Algeria (e-mail: alidrandco@yahoo.fr)

M.T. Kimour is with LASE Laboratory, Badji Mokhtar-Annaba university, Algeria (e-mail: mtkimour@hotmail.fr)

The novelty of our approach is the application of genetic algorithms to the feature model of the system to determine what features the system needs to incorporate to the running configuration to make it optimal in regard of specific runtime environmental conditions.

The benefit of our approach is double; in one hand, reconfiguration plans are not written in advance, discharging developers from the effort of prescribing reconfiguration plans for every situation warranting reconfiguration and handling unforeseen situations. In the other hand, the modeling effort made at design time (as represented by the feature model of the system) is not only useful for producing the system but also provides a richer semantic base for autonomic behavior during execution. [6]

For example, let's consider the case of a smart-phone. Depending on the running context, the adaptability logic aims at balancing diverging objectives (such as battery level, quick response to user interactions and higher connectivity) by managing a set of active/inactive components (such as a 3G connector or GPS) and a set of parameters (such as screen brightness or the number of active cores in the CPU).

It appears that it is difficult to determine at design-time the settings that optimize the system objectives especially if the user requirements are to evolve (for example, if the battery level drops too much, reducing the electrical consumption becomes more important which influences the adaptation strategy). This brings us to argue that writing adaptability logic at design time is inefficient and motivates our present work.

The remainder of the article is organized as follows: Section 2 and 3 recalls two topics fundamental to this paper, namely, feature models and genetic algorithms. Section 4 introduces our proposed approach for deriving adaptation logic by the genetic computation on feature models. Related works are reviewed in section 5. Finally we conclude our work in section 6.

II. Feature models

A software feature is "a distinguishing characteristic of a software item" [7]. A feature model is a hierarchically arranged set of features, the relationships among these features that determine the composition rules and the cross-tree constraints, and some additional information, such as trade-offs, rationale, and justifications for features selection [8].

Figure 1 depicts a possible (much simplified) feature model of the product Line in the domain of smart-phones product family. This example is partially inspired by [9].

Features are hierarchically linked in a tree-like structure through variability relationships such as optional, mandatory, single-choice, and multiple-choice but also the cross-tree relations (i.e., requires, excludes, and cross-tree constraints). [9]

For instance, all smart-phones need to include the two components: hardware and software. Hardware includes a processor, a screen, a RAM memory and sometimes a 3G connector and a GPS. Software would imply an OS whether Windows-Phone or a Linux, a set of applications amount of a specific control application that can be a Win32 (that requires running on Windows) or a Posix application (that requires running on Linux).

Attributes can be associated to features. For example, the screen is characterized by it brightness, the RAM by its size, the GPS by its speed…

Finally, the blue boxes in figure 1 represent the current features, while the orange boxes represent potential variants that may be activated in the future.

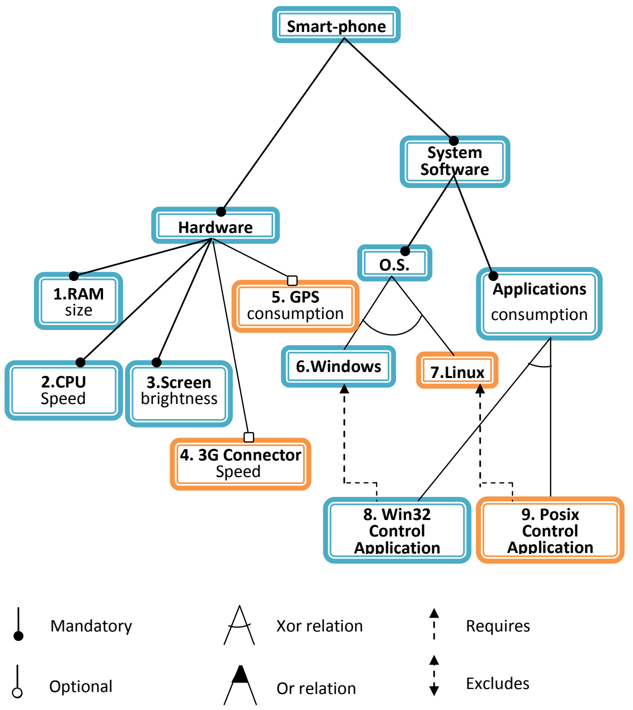

Figure1 : A (simplified) feature model for the domain of Smart-phones

For more details on FM notations semantic, reader can refer to [8,10,11,12]

The reason we base our approach on feature modeling is that the feature model represents somehow a repository of all the valid configurations of the system. As so, the feature model can be viewed as the search landscape of the system's optimization problem as we will explain it later on. Besides it is today well established that feature modeling has good tool support for automatic reasoning and verification[13] as it has been shown for example in [6]

III. Genetic Algorithms

Genetic algorithms are "stochastic-based search techniques that comprise a population of individuals, where each individual encodes a candidate solution in a chromosome" [14]. They are inspired by biological evolution of chromosomes. This includes mutation, recombination, and selection [15,16]. The main idea behind genetic algorithms is to gradually evolve an initial set of (possibly random) solutions for an optimization problem, to newer ones in a way that their fitness is improved from generation to generation.

Genetic algorithms use crossover to exchange building blocks between two fit individuals, hopefully producing off-springs with higher fitness values. The two most common forms of crossover in genetic algorithms are one-point and two-points crossover. In one-point crossover, a position in the chromosome is selected at random and the parts of two parents after the crossover position are exchanged. In two-point crossover, two positions are chosen at random and the segments between them are exchanged.

While crossover aims at converging at (local) optima, the role of mutation is to introduce genetic variation that may have been lost through-out the population as a result of the crossover operator [16]. Mutation takes an individual and randomly changes parts of its encoded solution based on some specified mutation rate.

In Genetic Algorithm a generation is a round of generating new chromosomes (referred to as children) from older chromosomes (referred to as parents). The choice of parent chromosomes is known as selection and is based on a fitness function that is used to evaluate the quality of the solutions. With this survival of the fittest strategy, Genetic Algorithms often converge to a (near) optimal solution for optimization problems.

Typically, the Genetic algorithm repeats the process of generating new chromosomes based on the previous chromosomes until a specific condition is verified. This might be: an acceptable set of answers are developed, a maximum number of generations is reached, the maximum computing time is consumed…

Many works have shown that genetic algorithms can be extremely efficient in exploring the search space and rapidly converging to good solutions for complex and highly nonlinear problems especially when combined with local search methods [18][17]. In the next section we explain how genetic algorithms can be efficiently applied to feature models in order to explore the search space of valid configurations to find the most relevant one given a certain execution context.

IV. Approach Overview

In the present article we assume the existence of a monitoring and a reconfiguration infrastructure at runtime. The monitoring infrastructure observes the system and its execution environment and reports the corresponding Data to the decision-making process. The decision-making process interprets the monitoring data and determines what modifications need to be brought to the system's configuration. The reconfiguration infrastructure effects the changes throughout the system through the use of an adaptation driver [1].

Thus we focus here on the decision-making process. Traditionally, decision-making was implemented through reconfiguration rules which are difficult to manage and maintain. At the opposite, our work aims at generating new optimal configurations of the running system. We employ Genetic Algorithms to search the possible configuration space of the feature model in order to select the best configurations considering specific environmental context and user expectations. To this end we will carry out the following steps:

1. Encoding the system configurations (feature combinations) using genetic chromosomes.
2. Defining specific evolution operators.
3. Defining validity constraints over solutions
4. Defining an evaluation and selection method of the new found solutions.

The outline of our approach is depicted in Figure 2 and will be explained throughout the few next sections.

A. *Solutions representation*

In our settings, we use a two chromosomes based representation of solutions:

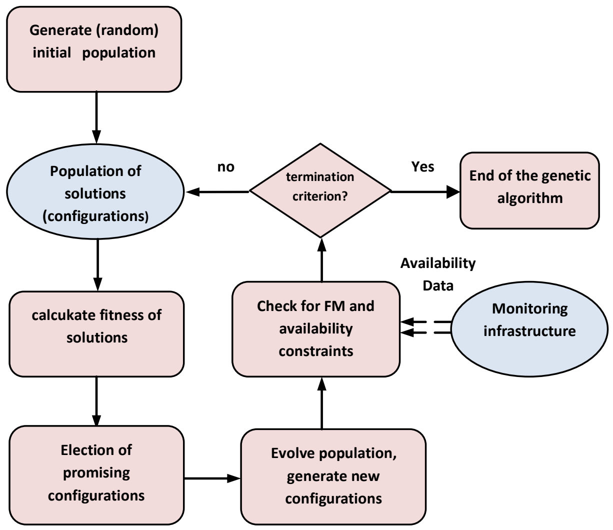
Generate (random) initial population
Population of solutions (configurations)
no
termination criterion?
Yes
End of the genetic algorithm
Availability Data
calcukate fitness of solutions
Check for FM and availability constraints
Monitoring infrastructure
Election of promising configurations
Evolve population, generate new configurations

Fig.2 overview of our approach

The first chromosome, we will designate as the *features chromosome* represents the active/inactive features in the feature model. That is, for a feature model of size *n* (i.e. a feature model with *n* features), a binary chromosomes of size n is created (9 in the case of feature model of figure 1). Each gene of the chromosome represents whether the corresponding feature is selected or not. For instance, considering the feature model of figure 1, the chromosome <1,1,1,0,0,1,0,1,0> Would represent a configuration incorporating a RAM (first features), a CPU second feature), a Screen(third feature), the Windows live operating system (sixth feature) and the win32 control application (eighth feature) but where features 3G connector, GPS, the Linux Os and control application are deactivated.

The second chromosome, we will designate as the parameters chromosome, regroups the values of the parameters of all the features. Every gene in the parameters chromosome encodes a real number value of a parameter of some feature. The length of the second chromosome is $\sum Pi$ where Pi is the number of the parameters of feature i. Specifically, in the case of our example. the parameters chromosome <128,3.1, 1, 3,2, 2,3> encodes a configuration where the RAM size equals 128MB, the processor speed equals 3.1Ghz, screen brightness equals 1, 3G Connector speed equals 3Mbits/s ; GPS power consumption equals 2w . the power consumption of the Win32 control application equals 2w and the power consumption of the Posix control application equals 3w.

The combination of the two chromosomes (the features and the parameters chromosomes) characterizes a potential system reconfiguration and is therefore considered to be a potential optimal solution. As we will explain it later, it is this combination of the chromosomes that serves to evaluate the new solutions fitness.

B. *Evolution operators*

The new chromosomes are generated based on the mutation and cross-over operators. Since we are using a two chromosomes representation, each one being structurally and semantically different, the mutation and cross-over operators are leveraged to allow the manipulation of the configurations using the double chromosomes representation. Specifically, we define two cross-over operators and two mutation operators. Each cross-over (and mutation) operator manipulates a different chromosome.

Given the structure of the features chromosomes, mutation causes one (or more) feature that is not present in the previous chromosome to be included in the configuration or vice-versa (one -or more- random gene be flipped in the chromosome); whereas mutation on the parameters chromosome causes the modification of some feature’s parameter. Therefore, mutation is quite useful in creating new configurations that are only slightly different from previous configurations (chromosomes).

Cross-over on the other hand, rotates the values of the genes around a central pivot gene in two chromosomes. When applied to the features chromosome, the cross-over “mixes” two sub-combinations of features. Whereas cross-over on the parameters chromosomes keeps the same active features but combines two configurations parameters. The resulting chromosomes are radically different from the original chromosomes since the values of many genes could possibly change in this process. This allows the genetic algorithms to diversify the research in the searching landscape.

It is important to mention that the genetic evolution of the second chromosome by the mutation operator, might affect insignificant genes, for example the genes corresponding to parameters within inactive features or those related to parameters that can't be modified online like the RAM size. In order to avoid this problem, a mask chromosome is used to determine what genes can be affected by the mutation operator.

C. *Constraints evaluation*

Genetic operators generate new individuals in a random way. This means that new found configuration may not be valid. Therefore, system constraints must be verified to guarantee the validity of new found configurations.

Validity constraints are explicitly expressed by the feature model of the system (mandatory, exclusive feature and global and cross-tree constraints) but can also vary depending on the available features. Extensive studies allow the automatic verification of the constraints of the feature model [8,9,10]. Availability constraints, in the other hand, need to be evaluated according to monitoring information reported by the monitoring infrastructure.

Since the genetic algorithm generates new solutions in a random way, there is a high probability for these solutions to be invalid. In order to avoid spending too much time in computing on invalid individuals, a correction method must be defined in order to transform invalid individuals to valid ones [17].

D. *Fitness functions*

Generally speaking, a single fitness function is not sufficient to quantify all possible effects of a particular reconfiguration when balancing multiple objectives [19]. Instead, a set of fitness functions should be defined to evaluate new configurations according to the optimization dimensions specified by end-users. To illustrate this point, we (simplistically) define two fitness functions for our current example; a power consumption function (noted: *PC*) and a connectivity function (noted: *Conn*). For the sake of simplicity let's assume that *PC* equals the sum of the power consumption of all active hardware features and Conn relates to the connectivity of the GPS and 3G connector. Let's assume finally that the power consumption of every feature depends is calculated separately, for example it is constant in the case of the GPS and RAM features but depends on the CPU speed for CPU and the 3G connector speed for the 3G connector feature.

We define an additional function, *user_conformance* that calculates the degree of conformance of a configuration to the user preferences and aims at preserving user choices as much as possible, for example, when the user chooses to activate the Linux operating system, the *user_conformance* function would prevent the adaptation system from deactivating the Linux Os and activating the Windows-Phone instead.

E. *Solution selection*

The evaluation of fitness functions produces a set of optimal solutions called the Pareto front [22]. In order to choose one single solution, a common method consists in associating weighting coefficients to each fitness function. Weighting coefficients determine the importance or relevance of fitness functions regarding other fitness functions. A weighted sum can be used to combine the values obtained from each fitness function into one scalar value. In our case, we roughly define the weighted sum as follows:

$$GF = \sigma 1 * (1/PC) + \sigma 2 * Conn + \sigma 3 * user\_conformance \quad (1)$$

Coefficients must be set judiciously to reflect user requirements. For example the coefficient of the *user_conformance* function *(σ3)* will naturally be affected the maximum value. However, as requirements are likely to change while the application executes, developers should introduce code to rescale the coefficients of individual fitness functions. In particular, the fitness landscape is shifted when the coefficients of a fitness function are rescaled. By updating the relevance of each fitness function at run time, the genetic algorithm will be able to evolve reconfiguration strategies that address changes in requirements and running conditions.

For example, if the battery charge drops under a certain value the system should automatically doubles the current coefficient for power consumption *(σ1)* to give more importance to battery saving. Note that although we describe how these coefficients should be rescaled in response to high-level monitoring events, we do not explicitly specify reconfiguration plans. That is, we do not prescribe which components of the smart phone will be deactivated or how each one modifies it's parameters to consume less battery.

V. RELATED WORKS

Various adaptability approaches has been proposed in the literature in a variety of contexts. In the next few lines we provide an overview of the most related ones to ours. Some approaches propose a complete platform that handles system monitoring, decision-making and reconfiguration. Here, we will only focus on decision-making which is the problem addressed in the present paper.

Feature model have been already combined with genetic algorithms in other contexts. For example, F. Ensan, E. Bagheri, and D. Gasevic [20] successfully combined feature models and genetic algorithms in the context of automatic test generation. The objective is to explore the configuration space of a software product line feature model in order to automatically generate test suites that optimize two fitness functions; errors coverage and features coverage. The authors claim that the proposed approach is able to generate test suites of O(n) size complexity as opposed to O(2n) while maintaining a good level of test suites quality.

Authors in [23] proposed an evolutionary based approach for features selection in the context of products configuration at design time. They identified the challenge of finding an optimized feature selection that minimizes or maximizes an objective function during the products derivation process. They introduced their algorithm called GAFES which is an adaptation of the classical GA for features selection. The authors also reported empirical results where they claimed that the their algorithm can produce solutions with 86-97% of the optimality of other automated feature selection algorithms and in 45-99% less time than existing exact and heuristic feature selection techniques [23]. The GAFES algorithm is somewhat similar to our proposal, however it does not deal with online but with offline variability. Hence, the algorithm uses a single chromosome individuals representation and does not deal with parameters selection but only with active/inactive features. Besides, our algorithm addresses multiobjectives problems while GAFES only uses a single objective function.

Like us, authors in [6] base their approach on the reuse of the feature models to achieve adaptability at runtime. The benefits are immediate, as the design knowledge and existing model-based technologies can be reused at runtime [21]. However, the reconfiguration rules are hand-written which makes rules management difficult and error prone especially in the case of highly complex systems.

In [19], the proposed approach uses the feature model to represent the possible evolutions of the system. The authors base their decision-making process on a fitness function to select the most relevant configuration given a certain context. In the opposite of our approach, they propose the evaluation of all valid configurations. The authors admit that this may require an important computational effort. Although the authors argue that restraining the evaluation to valid configurations only reduces the computational effort considering the feature model constraints, we still think that the approach presents a scalability problem in presence of highly complex systems. Moreover, the approach does not deal with multi objectives problems.

Other works, use standard and advanced control solutions to tackle the problem of decision making. For example, [25] proposes in the context of web application servers, two proportional integral controllers to guarantee proportional delay differentiation and absolute delay in the database connection pool. [26] exploits the idea of model identification in order to adjust the CPU percentage dedicated to the execution of a web server. The identification is achieved via a first-order auto-regressive model based on a proportional integral control structure together with an adaptive controller. The recursive least squares method is used to estimate the model parameters.

Ramirez et al [1] proposed the use of genetic algorithms to automate the process of decision making of an autonomous system. They successfully applied their approach to the dynamic reconfiguration of a collection of remote data mirrors, with the goal of minimizing costs while maximizing data reliability and network performance.

Bitirgen et al. used Artificial Neural Network to predict the application performance from sensor data (cache space, offchip bandwidth…)[27]. A training phase is necessary and the training Data is often crucial. Here the training phase is done online allowing the decision making process to be accurate even in the presence of changing operating conditions.

Our main contribution regarding these approaches is the reuse of the feature model the system. The feature model helps writing the model of the word and offers a formal framework allowing automatic reasoning and verification of new found configurations.

VI. Conclusion and perspectives

Increasingly, pervasive and autonomic systems are present in our lives. For this kind of systems little effort is left at the user side, adaptability is a major capability that needs to be implemented. At the same time, increasing system complexity raises the challenge of maintaining adaptability logic. In response, adaptability logic derivation must be automated to avoid errors and inconsistencies.

In this paper we present a new approach that leverages the feature model of the system to achieve adaptability at runtime. The novelty of our approach is the use of evolutionary algorithms to calculate target configurations to optimize system objectives with respect to evolving environmental context. By basing our approach on the feature model we aim at reusing the design knowledge and existing model-based technologies at runtime. GA in the other hand allows us to discharge the developers from writing prescriptive rules to address particular scenarios warranting reconfiguration. Instead, our proposed approach is able to evolve reconfiguration plans to address situations that were not anticipated at design time, by incorporating system and environmental monitoring information with a genetic algorithm. Furthermore, reconfiguration plans can be evolved in response to changing requirements and environmental conditions by rescaling individual fitness functions. Lastly the Pareto approach adopted allows a better coverage of the front of optimal solutions and handles naturally the expression of minimum value of certain objectives.

Currently, the implementation of a tool that supports the proposed approach is a work in progress. In addition, we intend applying the proposal to realistic systems to prove its feasibility and effectiveness. Finally, our future work includes defining a technique that automatically translates the feature model of a system in the corresponding genetic algorithm that implements the decision-making process.

References

[1] Andres J. Ramirez, David B. Knoester, Betty H.C. Cheng, Philip K. McKinley. Applying Genetic Algorithms to Decision Making in Autonomic Computing Systems, ICAC'09, June 15–19, 2009, Barcelona, Spain. ACM

[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, New York, NY, USA, 2006. ACM.

[3] 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, Berlin, Heidelberg, 2008. Springer-Verlag

[4] 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..

[5] 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, Washington, DC, USA, 2004. IEEE Computer Society.

[6] Carlos Cetina, Pau Giner, Joan Fons, and Vicente Pelechano, "Autonomic Computing through Reuse of Variability Models at Runtime: The Case of Smart Homes", IEEE Computer, 2009, pp 46-52.

[7] IEEE Std 829-1998. "IEEE Standard for Software Test documentation". 16 September 1998.

[8] A. S. Karata¸s, H. Oguztüzün, and A. Dogru. "Global Constraints on Feature Models". Proceedings of Principles and Practice of Constraint Programming - 16th International Conference (CP-2010), Scotland 2010. Springer, vol. 6308, pp. 537-551. ISBN 9783642153952.

[9] Carlos Eduardo Alvarez Divo, Automated Reasoning on Feature Models via Constraint Programming, master thesis., June 2011

[10] A. S. Karata¸s, H. Oguztüzün, and A. Dogru. "Mapping Extended Feature Models to Constraint Logic Programming over Finite Domains". Proceedings of Software Product Lines: Going Beyond - 14th International Conference, (SPLC-2010), South Korea 2010. Springer, vol. 6287, pp. 286-299. ISBN 9783642155789.

[11] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortes. Automated reasoning on feature models. In 17TH Conference Advanced Information Systems Engineering, Springer, 2005.

[12] D. Benavides. "On The Automated Analysis of Software Product Lines using Feature Models. A framework for developing automated tool support". Sevilla, May 2007.

[13] D. Benavides, P. Trinidad, and A. Ruiz-Cortés, "Automated Reasoning on Feature Models," Proc. 17th Int'l Conf. Ad-vanced Information Systems Eng. (CAiSE 05), LNCS 3520, Springer-Verlag, 2005, pp. 491-503

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

[15] Faezeh Ensan, Ebrahim Bagheri, Dragan Gasevic: Evolutionary Search-Based Test Generation for Software Product Line Feature Models. CAiSE 2012: 613-628

[16] K. De Jong, "An analysis of the behavior of a class of genetic adaptive systems," Doctoral Dissertation. Ann Arbor: The University of Michigan, 1975.

[17] Tarek A. El-Mihoub, Adrian A. Hopgood, Lars Nolle, Alan Battersby. Hybrid Genetic Algorithms: A Review in Engineering Letters, 13:2, EL_13_2_11. August 2006

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

[19] Mohammad Ullah Khan, Roland Reichle and Kurt Geihs, Applying Architectural Constraints in the Modeling of Self-adaptive Component based Applications, the workshop Model-driven Software Adaptation (M-ADAPT'07), July 2007.

[20] Faezeh Ensan, Ebrahim Bagheri, Dragan Gasevic: Evolutionary Search-Based Test Generation for Software Product Line Feature Models. CAiSE 2012: 613-628

[21] Nelly Bencomo, Jaejoon Lee, Svein O. Hallsteinsen, "How dynamic is your Dynamic Software Product Line?". In proceeding of: Software Product Lines - 14th International Conference, SPLC 2010, Jeju Island, South Korea, September 13-17, 2010. Workshop Proceedings (Volume 2 : Workshops, Industrial Track, Doctoral Symposium, Demonstrations and Tools).

[22] P. Ngatchou, Anahita Zarei, "Pareto Multi Objective Optimization". In Intelligent Systems Application to Power Systems, 2005. Proceedings of the 13th International Conference. Page(s): 84 – 91. ISBN: 1-59975-174-7

[23] Jianmei Guo, Jules White , Guangxin Wang , Jian Li , Yinglin Wang. "A Genetic Algorithm for Optimized Feature Selection with Resource Constraints in Software Product Lines" in the Journal of Systems and Software, February, 2011

[24] W. Pan, D. Mu, H. Wu, and L. Yao. Feedback Control-Based QoS Guarantees in Web Application Servers. In Proceedings of the 10th International Conference on High Performance Computing and Communications, pages 328–334, September 2008.

[25] X. Liu, X. Zhu, S. Singhal, and M. Arlitt. Adaptive entitlement control of resource containers on shared servers. In Proceeding of the 9th IFIP/IEEE International Symposium on Integrated Network Management, pages 163–176, May 2005.

[26] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing energy and server resources in hosting centers. In Proceedings of the eighteenth ACM symposium on Operating systems principles, SOSP '01, pages 103–116, New York, NY, USA, 2001. ACM.