2016 IEEE 24th International Requirements Engineering Conference

The Generation and Evolution of Adaptation Rules in Requirements Driven Self-adaptive Systems

Tianqi Zhao
Key Laboratory of High Confidence Software Technology, Ministry of Education, China
Institute of Software, School of EECS, Peking University, Beijing, 100871, China
{zhaotq12}@sei.pku.edu.cn

***Abstract*—One of the challenges in self-adaptive software systems is to make adaptation plans in response to possible changes. A good *plan* mechanism shall have the capability of: 1) selecting the most appropriate adaptation actions in response to changes both in the environment and requirements; 2) making adaptation decisions efficiently to react timely to arising situations at run-time. In existing approaches for *plan* process, rule-based adaptation provides an efficient offline planning method. However, it can react neither to changeable requirements nor to unexpected environment changes. On the contrary, goal-based and utility-based approaches provide online planning mechanisms, which can well handle a highly uncertain environment with dynamically changing requirements and environment. However, online adaptation decision making is often computationally expensive and may encounter less-efficiency problems. The aim of our research is to improve the planning process in requirements driven self-adaptive systems, i.e., enabling the self-adaptive system to efficiently make adaptation plans to cope with the dynamic environment and changeable requirements. To achieve such advantages, we propose a solution to enhance the traditional rule-based adaptation with a rule generation and a rule evolution process, so that the proposed approach can maintain the advantages of efficient planning process while being enhanced with the capability of dealing with runtime uncertainty.**

***Index Terms*—requirement driven self-adaptation, adaptation plan, reinforcement learning, case-based reasoning**

I. Introduction

Modern software systems usually execute in an environment with high uncertainty, in which, not only the environment attributes but also the requirements might change unexpectedly. Self-adaptive system provides a promising solution to tackle such uncertainty, and in particularly, enabling the system to react to changes both in the environment and requirements ([1], [2]).

One of the challenges in self-adaptive software system is effective *planning*, i.e., selecting the most appropriate adaptation actions when changes occur. A good *plan* mechanism should have the capability of: 1) making effective adaptation decisions, i.e., selecting adaptation actions in respond to changes both in the environment and requirements, and the execution of selected adaptation action shall guarantee the satisfaction of requirements; 2) making adaptation decisions efficiently, i.e., the planning process should be efficient so that a self-adaptive system can react timely to arising situations at run-time.

Generally, there are three different types of *planning* methods, i.e., rule-based planning, goal-based planning and utility function-based planning. While rule-based planning ([3], [4], [5]) is an offline *planning* method that selects adaptation actions according to prescribed rules, the goal-based ([6], [7]) and utility function-based planning ([8], [9]) are online planning methods that leave the system the task to reason about the most appropriate adaptation actions.

The offline planning methods have the advantages of efficient planning process since its adaptation logic is prescribed at the offline phase. However, it is less likely to find the optimal adaptation solutions since: 1) the traditional adaptation rules can only react to changes in environment attributes, while the changes in requirements are beyond its scope; 2) the rules prescribed at the offline phase cannot well predict the online conditions and will result in poor performance when the system in dynamic environment does not behave as expected. On the contrary, the online planning methods are more likely to find the optimal adaptation solutions, since they take the runtime environment and the requirements into consideration when searching for the most appropriate adaptation actions in the action space. However, searching for optimal decisions is often computationally expensive and encounters less-efficiency problems ([2], [8], [10]).

The aim of our research is to improve the *planning* process in requirements driven self-adaptive systems, i.e., enabling the self-adaptive system to efficiently make adaptation plans to cope with the dynamic environment and changeable requirements.

The remainder of the paper is organized as follows. Section II introduces our research question and Section III highlights the challenges. Section IV describes the related works and the state-of-art of the research question. Section V introduces proposed solutions that can tackle the challenges and improve the state-of-art. Finally Section VI introduces the progress and Section VII concludes the contributions.

II. Research questions

This research focuses on the *planning* process in requirements driven self-adaptive systems. The aim of our research is to design an improved *planning* mechanism, which can combine the advantages of both online and offline methods, i.e., combining both the effectiveness of planning results and the efficiency of planning process. Here the "effectiveness" means that: 1) the planning process can react to both the changes in environment and requirements; 2) the planning results can guarantee the satisfaction of requirements; 3) the planning

2332-6441/16 $31.00 © 2016 IEEE
DOI 10.1109/RE.2016.18

IEEE computer society

RE 2016, Beijing, China
Doctoral Symposium Paper

mechanism can well utilize the real feedback information to improve its adaptation logic.

To achieve such advantages, we propose a solution to enhance the traditional rule-based adaptation with a rule generation and a rule evolution process. The objectives of this work are to:

1) design a rule generation mechanism to automatically generate optimal or near-optimal rules from requirements;
2) design a rule evolution mechanism that can evolve rules from real feedback information to cope with the dynamic environment, and efficiently learn a new set of updated rules in response to the changeable requirements;

These objectives can improve the traditional *planning* methods both in effectiveness and in efficiency. On one hand, the planning process is reduced to the very efficient rule-based planning; on the other hand, it can be guaranteed that rules can always adapt themselves with dynamic environment and changed requirements.

III. Challenges

We have identified the following challenges to achieve our research objectives:

- **The selection of learning algorithm.** The learning algorithm shall automatically generate a set of rules that can lead to desirable adaptation results and guarantee the satisfaction of requirements. A challenge is to design an effective learning algorithm that can generate rules from requirements, i.e., an algorithm that can generate rules by taking into consideration how the software and environment can interact with each other to affect the requirements. Therefore, we may need to find an appropriate machine learning algorithm, and then fit it for the situation of self-adaptive systems.
- **The design of efficient evolution algorithm.** The rule evolution algorithm shall learn a new set of updated rules when requirements change. This evolution process shall be efficient enough to react to the new arising situations timely. However, it usually takes time for a learning algorithm to conduct and converge, so it is not easy to update the rule set in respond to requirements change timely. Therefore, we may need to design an incremental learning method or transfer learning method to facilitate the efficient evolution, i.e., using experience gained in learning to perform one task to help improve learning performance in a related but different task.
- **The utilization of real feedback information.** The rule evolution algorithm shall utilize the real feedback information to evolve the learnt rules. Such evolution process shall guarantee that the evolved rules cope with runtime environment. To this end, we should consider what kinds of feedback information shall be observed, and how to use the observed feedback to enhance the rules.

IV. Related Works

We will introduce some related works in the *planning* methods of requirements driven self-adaptive systems.

A. Offline Planning Methods

Rule-based planning provides a powerful offline planning mechanism, in which, a set of rules are prescribed to specify adaptation logic of what particular actions should be performed to react to monitored events. Typically, an adaptation rule takes the form of "IF (condition) THEN (action)", where the condition specifies a trigger for the rule, and the action specifies an operation sequence to perform in response to the trigger. For example, *Rainbow* [4] uses rules to specify the desired adaptive behavior; Wang et.al [11] proposes a rule-based framework for self-adaptation with the consideration of the separation between the application and the adaptation logic; and Acher et.al [5], [12] use adaptive rules to configure the adaptive system with respect to a particular context. Besides such "condition-action" rules, the adaptation rules can also be specified by a state transition diagram [13], where the nodes in the diagram specify different system configurations and the edges specify the transition trigger from the source to target nodes.

Obviously, rule-based adaptation has the advantages of efficient planning process, since it does not need a online reasoning process. However, the traditional rule-based adaptation has two limitations: 1) Adaptation rules are often specified offline with insufficient knowledge about execution scenario, so the rules will result in poor performance when the system in dynamic environment does not behave as expected; 2) Rule-based adaptation can only react to environment changes, while the changes in goals and software behavior patterns are beyond its scope.

B. Online Planning Methods

Goal-based planning reduces the adaptation decision making as a satisfiability problem, where the planner searches the action space for optimal actions that contribute to the satisfaction of requirement goals. The requirements goals include functional goals that underlie services the system is expected to deliver, and non-functional goals that refer to expected system qualities such as security, safety, performance and usability [14]. In a self-adaptive system, the functional goals can be achieved by alternative software configurations, while the non-functional goal can be referred as criterion to trade off the alternative configurations [15], i.e., the configurations that contribute to the high-priority non-functional goals are regarded as optimal solutions. Some recent works make improvements on goal-based planning methods. Chen et.al [6] propose to integrate goal-based adaptation with architecture-based adaptation, while Qian et.al [9] propose to integrate goal-based adaptation with case-based adaptation. Filieri et.al ([16], [17]) apply control theory to goal-based planning. G.Ghezzi et.al ([18], [7]) propose to make adaptation decisions through probabilistic model checking to maximize the system's ability to meet its non-functional requirements. G.A.Moreno et.al [19] present a proactive adaptation approach that reasons about the adaptation decision based on a look-ahead horizon for the achievement of requirements. D'Ippolito et.al [20] propose to categorize the

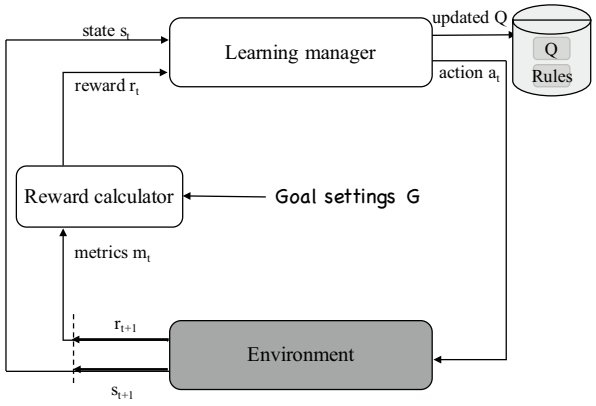

Fig. 1: Generate Adaptation Rules Through Reinforcement Learning

idealization of environment into several levels, where different requirements goals shall be satisfied under different idealization levels.

Utility-based planning reduces the adaptation decision making as an optimization problem, where the utility function maps a system configurations to a real value that indicates the desirability of this configuration, and the planner attempts to select actions to maximize the utility function ([21], [22], [23]). There are works that make improvement on the utility-based planning. Elkhodary et.al [24] proposes a learning based feature-oriented self-adaptation framework FUSION that enhances the planning process with a learning cycle to better the planning accuracy. Bencomo et.al [25] propose to use dynamic decision networks to support utility based planning, while Pasquale et.al [26] propose to use causal networks for planning. Some works ([27], [28], [29]) use reinforcement learning methods to help online decision making for the maximize of the aggregated utility values in the long run.

Goal-based and Utility-based methods are both online planning methods, in the sense that the adaptation plan is not prescribed by developers at design time but is dynamically derived by the system at runtime. Compared to rule-based planning, the online planning methods are more likely to find the optimal adaptation solutions. However, searching for optimal decisions is often computationally expensive and encounters less-efficiency problems.

V. Solutions

To achieve the research goals highlighted in Section II and tackle the challenges in Section III, I consider the following methodologies.

A. Rule Generation Algorithm

1) Reinforcement learning: In a general reinforcement learning problem [30], an agent interacts with the environment in the following ways:

- At time t, the agent receives the current state s_t, based on which the agent selects an action a_t and then executes it.
- At time $t+1$, the agent finds itself in a new state s_{t+1}. As a consequence of its action, a reward r_{t+1} will be observed.

Roughly speaking, solving a reinforcement learning problem is to find the optimal policy that can maximize the observed *reward* in the long run.

2) Fit Reinforcement learning for Rule Generation: I decide to apply the reinforcement learning method to the generation of adaptation rules due to the following reasons:

- an adaptation rule specifies the optimal action in a specific condition, while the reinforcement learning algorithms aim to learn an optimal policy that defines the best action to take in a specific state;
- the reinforcement learning method can learn from the interaction of the agent with the environment, and we also want to derive adaptation rules from the interaction of the software with the environment;
- the reinforcement learning method can be further used to continuously reinforce the learnt rules at runtime;

A reinforcement learning-based method (as depicted in Figure 1) has been designed to support rule generation, where the rule can be generated from a goal setting as follows.

1. The reinforcement learning method uses the given goal setting to tune the reward function. A goal setting here is a weight vector that indicates the priority of goals, and can be denoted as

   $$G =< \omega_1, ..., \omega_n >$$

   where n is the number of goals, and ω_i indicates the priority of *ith* goal. Here a larger weight indicates a higher priority, and the goal with a higher priority can be satisfied at the expense of the goals with a lower priority. The tuned reward function is

   $$r_t = s_t^1 \times \omega_1 + ... + s_t^n \times \omega_n$$

   where s_t^i is the satisfaction degree of goal g_i at time t and can be calculated from a goal-related metric using an utility function.
2. With the tuned reward function, the learning manager can interact with the environment to derive a set of adaptation rules. The interaction is conducted continuously, where the learning manager selects actions and then the environment respond to those actions by presenting new situations. Through such interactions, the quality function Q is continuously being updated. When the learning algorithm converges, a set of adaptation rules is derived from the final Q, in which, each rule specifies the optimal action to take given the goal setting G in a specific state.

Here the quality function Q indicates how good is it to take a specific condition in a specific action. Let $Q(s, a)$ denotes the quality value of taking action a under state s, then the optimal action in state s (denoted as $A^*(s)$) is the one that can yield the largest quality value, i.e.,

$$A^*(s) = argmax_{a \in A(s)} Q(s, a)$$

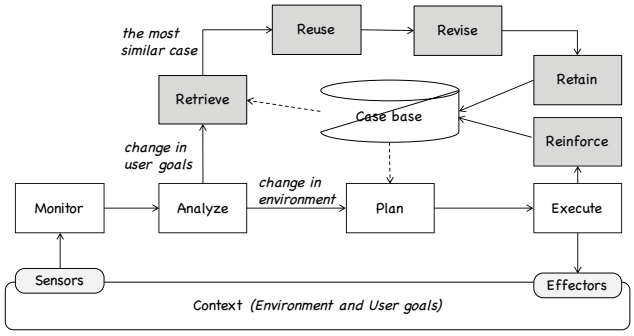

Fig. 2: Evolve Adaptation Rules Through Case-based Reasoning

where $A(s)$ is the set of legal actions in state s. Therefore, with the learnt quality function, an optimal rule can be derived as "IF s THEN $A^*(s)$".

B. Rule Evolution Algorithm

1) Case-based Reasoning: Generally, case-based reasoning [31] is a problem solving paradigm that utilizes the specific knowledge of previously experienced problems to solve a new problem. The new problem can be solved by finding and reusing a most similar old problem.

A general case-based reasoning cycle can be described by the following four processes: *retrieve* the most similar cases; *reuse* the knowledge in the most similar cases to form a solution for the new problem; *revise* the formed solution; *retain* the parts of this experience for future problem solving.

2) Fit Case-based Reasoning for Rule Evolution: I adopt case-based reasoning to support the rule evolution process reacting to the changes of user goals, and have designed a case-based evolution algorithm that contains a two-phase process:

- at the offline learning phase, a case base will be built for a collection of goal settings, where each case includes an optimal or near-optimal rule set for a specific goal setting;
- at the online phase, rule-based adaptation is enhanced with the rule evolution activity, in which, the best fitting case is retrieved from the case base according to the current goal setting;

By combining such a two-phase process, we can shift as much of the learning cost as possible to the offline phase and bring only case-based retrieve activity to runtime.

At the offline phase, the case base will be built for a collection of "representative" goal settings, where a case in the base consists of a goal setting and the learnt rules that is cope with this goal setting. Here "Representative" means that the goal setting collections shall cover a wide range of possible priority settings, for example, there is the goal setting that assigns the goals equal priorities, and also the goal setting that assigns one of the goal the highest priority.

At the online phase, case-based reasoning will be performed when the user goal setting changes, where the case-based reasoning cycle includes the following activities as shown in Figure 2:

- **Retrieve** the most similar case for the new goal setting. The most similar case is the case whose goal setting has the smallest distance from the new goal setting. Here the distance between two goal settings G_A and G_B are defined as their euclidian distance, i.e.,

$$distance(G_A, G_B) = [(g_1^A - g_1^B)^2 + ...(g_n^A - g_n^B)]^{\frac{1}{2}}$$

- **Reuse** the adaptation rules in the most similar case. The rules in the most similar case will be reused to support the further rule-based adaptation. I've made the hypothesis that a smaller case distance can yield a larger rule set similarity. I've tested this hypothesis through experiment, which proves it positively.
- **Revise** the reused rules if necessary. The learnt rules will be continuously improved at runtime by the feedback information.
- **Retain** the gained experience. The case base will be updated in the sense that an existing case might be updated or a new case might be added to the case base.

With case-based reasoning, a best fitting case can be retrieved and reused when the runtime goal setting changes. That is, the case of G_i will be used when the current goal setting is set to G_i. If the existing case does not exist, a most similar case will be reused instead.

In this way, the rule-based adaptation is able to react to not only the environment changes but also the changes of requirement goals:

- When the goal setting changes, case-based reasoning will be triggered to retrieve a best fitting case for the new goal setting. The rule set of the new goal setting will be activated to support the rule-based adaptation.
- When environment changes, the rule-based adaptation will be performed according to the current activated rules, i.e., the action of a rule will be executed when its condition is triggered by the environment changes.

3) Evolution based on Feedback information.: The execution of an adaptation action might affect the metrics of the system, e.g., adding a server might seed up the "response time" while increasing the "cost". Such feedback information can be utilized to decide the effectiveness of adaptation actions, and therefore evolve the adaptation rules when necessary. That is, if the execution of an adaptation action improves the system metrics, then a positive reward shall be gained indicating the effectiveness of this adaptation action. On the other hand, if the system metrics become worse after adaptation, then a negative reward shall be gained as a punishment. When the original optimal adaptation action is not optimal anymore according to such feedback information, the corresponding adaptation rule will be evolved accordingly.

Such evolution can be implemented through reinforcement learning methods, in the sense that its quality function can be updated after an adaptation using the arising rewards, which can be calculated from the feedback metrics. When the changes of quality function are significant enough to derive different adaptation rules, the rule set will be updated accordingly.

VI. Progress

By now I have finished the design of algorithms, including the rule generation algorithm and the rule evolution algorithm.

I plan to evaluate this research based on the case study of an e-commerce website, which can adapt its components including verification type and browse mode in respond to the environment changes including workload and concurrent users as well as to the changeable goals including "quick response", "high security", and "high usability".

I intend to evaluate this research by answering three research questions:

1). To what extent does the rule generation achieve the user-defined goals?
2). To what extent does the rule evolution algorithm support the rule evolution in response to changing goals?
3). How well does the evolution algorithm utilize the real feedback and reinforce the rule set?

To answer the research question 1, I plan to apply the rule generation algorithm to learn rules from a specific goal setting, and then uses the generated rules to support rule-based adaptation given this goal setting.The goal satisfaction degrees during the adaptation process will be recorded, and high satisfaction degrees can give a positive answer to research question 1. To answer the research question 2, I plan to apply the rule evolution algorithm to a scenario where the goals might change dynamically, i.e., new goals might be added and the goal priorities might be changed, and test whether the rule evolution algorithm can update the rule set according to the changeable goals. To answer the research question 3, I plan to apply the rule evolution algorithm to a scenario where the software behavior pattern might change dynamically, i.e., the way how the goals are affected by the software and the environment might change dynamically, and test whether the rule evolution algorithm can evolve the rules using real feedback information to cope with such uncertainty.

Moreover, I plan to compare this work with the existing approaches, including the traditional goal-based adaptation and the traditional rule-based adaptation approaches.

VII. Contributions

This research can facilitate the filed of requirements driven self-adaptation. It focuses on the *planning* process of requirements driven self-adaptation, and provides an improved *planning* mechanism that can combine the advantages of the state-of-art approaches.

The general *planning* methods can be categorized into online planning methods and offline planning methods. However, the former methods can make effective adaptation decisions but often encounter less-efficiency problems, while the latter methods have a efficient planning process but cannot make good adaptation decisions in a highly uncertain environment.

The proposed method combines the advantages of both approaches by enhancing the traditional rule-based adaptation with a rule generation and rule evolution process. In this way, it can enjoy the advantages of rule-based adaptation while being enhanced with the capability to deal with runtime uncertainty.

- The proposed method has the advantages of efficient planning process, since it makes adaptation decisions according to adaptation rules. It is still a rule-based adaptation method otherwise that the rules are learnt from goals instead of being prescribed by developers.
- The proposed method can make effective adaptation decisions, in the sense that: 1) it can react to not only the environment changes but also the requirement changes through case-based reasoning. A best fitting rule set will be retrieved and reused when the goal setting changes, which can guarantee that the activated rule set always copes with goals; 2) it can evolve the rules from real feedback information, so that the rules can always be adapted to cope with the runtime situation.

VIII. Related Publications

Tianqi Zhao, Haiyan Zhao, Wei Zhang. A preliminary study on requirements modeling methods for self-adaptive software systems[C]// Asia-Pacific Symposium on Internetware. ACM, 2013:1-10.

Tianqi Zhao, Haiyan Zhao, Wei Zhang, Zhi Jin. User preference based autonomic generation of self-adaptive rules[C]// Asia-Pacific Symposium on Internetware on Internetware. ACM, 2014:25-34.

Tianqi Zhao, Haiyan Zhao, Wei Zhang, Zhi Jin. Goal model driven alternative selection: a quantitative approach[C]// International Model-Driven Requirements Engineering Workshop. 2015:1-10.

IX. Acknowledgements

This research was supported by the National Basic Research Program of China (the 973 Program) under grant 2015CB352201 and the National Natural Science Foundation of China under grants 91318301, 61432020 and 61272163.

References

[1] B. H. Cheng and et al, "Software engineering for self-adaptive systems: A research roadmap," in *Software Engineering for Self-Adaptive Systems* (B. H. Cheng, R. d. Lemos, H. Giese, P. Inverardi, and J. Magee, eds.), pp. 1–26, Springer-Verlag, 2009.

[2] R. de Lemos, H. Giese, H. A. Müller, and et. al., "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II - International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*, pp. 1–32, 2010.

[3] I. Lanese, A. Bucchiarone, and F. Montesi, "A framework for rule-based dynamic adaptation," in *Proceedings of the 5th International Conference on Trustworthly Global Computing*, TGC'10, pp. 284–300, Springer-Verlag, 2010.

[4] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *IEEE Computer*, vol. 37, no. 10, pp. 46–54, 2004.

[5] M. Acher, P. Collet, F. Fleurey, P. Lahire, S. Moisan, J.-P. Rigault, and et al., "Modeling context and dynamic adaptations with feature models," in *Proceedings of the 4th International Workshop on Models@run.time*, 2009.

[6] B. Chen, X. Peng, Y. Yu, B. Nuseibeh, and W. Zhao, "Self-adaptation through incremental generative model transformations at runtime," in *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, pp. 676–687, 2014.

[7] C. Ghezzi, L. S. Pinto, P. Spoletini, and G. Tamburrelli, "Managing non-functional uncertainty via model-driven adaptivity," in *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, pp. 33–42, 2013.

[8] N. Esfahani, A. M. Elkhodary, and S. Malek, "A learning-based framework for engineering feature-oriented self-adaptive software systems," *IEEE Trans. Software Eng.*, vol. 39, no. 11, pp. 1467–1493, 2013.

[9] W. Qian, X. Peng, B. Chen, J. Mylopoulos, H. Wang, and W. Zhao, "Rationalism with a dose of empiricism: Case-based reasoning for requirements-driven self-adaptation," in *IEEE 22nd International Requirements Engineering Conference, RE 2014, Karlskrona, Sweden, August 25-29, 2014*, pp. 113–122, 2014.

[10] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 4, no. 2, p. 14, 2009.

[11] I. Lanese, A. Bucchiarone, and F. Montesi, "A framework for rule-based dynamic adaptation," in *TCG 2010: Lecture Notes on Computer Science 6084* (M. Wirsing, M. Hofmann, and A. Rauschmayer, eds.), pp. 284–300, Springer-Verlag Berlin Heidelberg, 2010.

[12] M. Acher, P. Collet, P. Lahire, S. Moisan, and J.-P. Rigault, "Modeling variability from requirements to runtime," in *Proceedings of 16th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, pp. 77–86, IEEE Computer Society, 2011.

[13] N. Bencomo, P. Sawyer, P. Grace, and G. S. Blair, "Dynamically adaptive systems are product lines too: Using model-driven techniques to capture dynamic variability of adaptive systems.," in *Software Product Lines, International Conference, Splc 2008, Limerick, Ireland, September 8-12, 2008, Proceedings. Second Volume*, pp. 23–32, 2008.

[14] v. A. Lamsweerde, "Goal-oriented requirements engineering: A guided tour," in *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, pp. 249–262, IEEE Computer Society, 2001.

[15] A. Lapouchnian, Y. Yu, S. Liaskos, and J. Mylopoulos, "Requirements-driven design of autonomic application software," in *Conference of the Center for Advanced Studies on Collaborative Research*, pp. 16–19, 2006.

[16] A. Filieri, H. Hoffmann, and M. Maggio, "Automated design of self-adaptive software with control-theoretical formal guarantees," in *Proceedings of the 36th International Conference on Software Engineering*, pp. 299–310, ACM, 2014.

[17] A. Filieri, H. Hoffmann, and M. Maggio, "Automated multi-objective control for self-adaptive software design," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE*, pp. 13–24, 2015.

[18] C. Ghezzi and A. M. Sharifloo, "Dealing with non-functional requirements for adaptive systems via dynamic software product-lines," in *Software Engineering for Self-Adaptive Systems*, pp. 191–213, 2010.

[19] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl, "Proactive self-adaptation under uncertainty: A probabilistic model checking approach," in *Proceedings of the Joint Meeting of the European Software Engineering Conference and the Symposium on Foundations of Software Engineering (ESEC/FSE)*, 2015.

[20] N. D'Ippolito, V. Braberman, J. Kramer, J. Magee, D. Sykes, and S. Uchitel, "Hope for the best, prepare for the worst: multi-tier control for adaptive systems," in *International Conference on Software Engineering*, 2014.

[21] D. Garlan and et al., "Rainbow: Architecture-based self-adaptation with reusable infrastructure," oct. 2004, pp. .," *IEEE Computer*, pp. 46–54, Oct 2004.

[22] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjorven, "Using architecture models for runtime adaptability," *IEEE Software*, vol. 23, no. 2, pp. 62–70, 2006.

[23] J. O. Kephart and R. Das, "Achieving self-management via utility functions," *IEEE Internet Computing*, vol. 11, no. 1, pp. 40–48, 2007.

[24] A. Elkhodary, N. Esfahani, and S. Malek, "Fusion: a framework for engineering self-tuning self-adaptive software systems," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pp. 7–16, ACM, 2010.

[25] N. Bencomo and A. Belaggoun, "Supporting decision-making for self-adaptive systems: from goal models to dynamic decision networks," in *Proceedings of the 19th international conference on Requirements Engineering: Foundation for Software Quality*, pp. 221–236, 2013.

[26] L. Pasquale, "Requirements-driven adaptive security: Protecting variable assets at runtime," in *Proceedings of the 2012 IEEE 20th International Requirements Engineering Conference (RE)*, pp. 111–120, 2012.

[27] D. Kim and S. Park, "Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software," in *International Workshop on Software Engineering for Adaptive and Self-managing Systems*, 2009.

[28] M. Amoui, M. Salehie, S. Mirarab, and L. Tahvildari, "Adaptive action selection in autonomic software using reinforcement learning," in *Autonomic and Autonomous Systems, 2008. ICAS 2008. Fourth International Conference on*, 2008.

[29] H. N. Ho and E. Lee, "Model-based reinforcement learning approach for planning in self-adaptive software system," in *International Conference on Ubiquitous Information Management and Communication*, 2015.

[30] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 1998.

[31] A. Aamodt and E. Plaza, "Case-based reasoning: foundational issues, methodological variations, and system approaches," *Ai Communications*, vol. 7, no. 1, pp. 39–59, 2009.