4DV610

Report

December 31, 2018

Semester: Autumn 2018
Authors: Sarpreet Singh Buttar
Prasannjeet Singh
Emails: {sb223ce, ps222vt}
@student.lnu.se

1 Introduction

Self-adaptation is one prominent approach that enables a software system to autonomously deal with uncertainties at runtime to achieve particular adaptation goals. Example of the uncertainties are changing user goals, dynamic availablity of the components, fluctuating conditions of the environment in which the software system operates, etc. Architecture-based adaptation is one of the common approach to relaize self-adaptation. It decomposes the system into two parts: a managed system that needs adaptation and contains the domain logic, and a managing system that holds the adaptation logic. These systems communicate with each other via sensors and actuators. The managing system achieves the adaptation goals of the managed system by using a feedback loop(s) that contains four actions: collect runtime data from the managed system and its environment, analyze the collected data, decide if an adaptation plan is required, and adapt the managed system according to the adaptation plan.

The field of self-adaptation has been continously studied over the last two decades by researchers and engineers from different fields. As a result, common knowlege has been established in the community. On the other hand, approaches that are in their early phases are still subject of debate. A recent paper [1] by Danny Weyns presents a tour of the field of self-adaptation during the last two decades. The tour is divided in to six waves of research. Each wave focuses on a key aspect that plays an important role in the software engineering of self-adaptive systems.

In this report, we apply the theory of the six waves to one of the state-of-the-art approach called ExecUtable RuntimE MegAmodels (EUREMA) [2]. Concretely, we find the key aspects mentioned by this approach and relate them to the appropriate wave. The remaining of this report is organized as follows. In section 2, we briefly explain the theory of six waves. Then in section 3, we apply the theory of six waves on EUREMA. Finally, we summarize this report in section 4.

2 Theory of Six Waves

In this section, we briefly explain the theory of six waves: automating tasks, architecture-based adaptation, runtime models, goal-driven adaptation, guarantees under uncertainties, and control-based adaptation.

2.1 Wave I - Automating Tasks

In 2003, IBM published a manifesto which targeted the rising complexity that system administrators faced while installing, mainting, configuring, etc., a software system. Therefore, the first wave focuses on automating the tasks of system administrators. Therefore, the idea of self-management is proposed in which a system manages itself autonomously, or with minimal human interference. Self-management is principally categorized into Self-configuration, Self-optimization, Self-healing and Self-protection. Self-configuration refers to the adaptation that allows a system to configure itself autonomously, whereas in self-optimization the systems constantly tries to improve its performance or cost. Self-healing allows the system to detect, diagnose and repair the problems that resulted in a failure. Lastly, self-protection enables the system to defend itself from malicious attacks or cascading failures. To realize self-management, IBM presented a MAPE-K reference model [3] that contains four different elements, namely Monitor, Analyze, Plan and Execute that share a common Knowledge element. The monitor element continously tracks the software system at runtime, the

analyze element analyze the data collected by the monitor. The plan element determines whether an adaptation is required, and if needed it creates an adaptation plan which then used by the execute element to adapt the software system. However, the concrete realization of the MAPE-K reference model embodies noteworthy challenges.

2.2 Wave II - Architecture-Based Adaptation

The approahes described in the first waves, e.g., MAPE-K reference model, provide solutions of self-adaptation. However, these solutions are based on a higher level of abstraction, hence they lack principled engineering perspectives on how to engineer self-adaptive systems. Therefore, this wave focuses on the foundational principles to engineer self-adaptive systems. Researchers aim to identify key concerns of self-adaptation by applying the principles of software architecture. As a result, a three-layer architecture model is presented by Kramer and Magge [4]. The bottom layer, i.e., component control, contains interconnected components that holds the functionality of the system, hence used to perform the adaptation. The middle layer, i.e., change management, comprises a set of pre-specified plans that are executed whenever a change is appeared in the bottom layer. The top layer, i.e., goal management, contains a blueprint of high-level goals, which is then used to produce a plan(s) whenever a request is received from the middle layer. This layer is also responsible for introducing new goals. Thus, the three-layer architecture model brings two fundamental architecture concerns named as change management that manages adaptation using plan, and goal management that generates plans based on high-level goals. These concerns present a systematic approach to manage the complexity of engineering self-adaptive systems. The approaches, including the three-layer architecture model, do not provide a clear vocabulary to define the primary architectural characteristics of self-adaptive systems. As a result, FOrmal Reference Model for Self-adaptation (FORMS) [5] is presented that builds on the traditional principles of self-adaptation and provides a precise vocabulary to define such characteristics.

2.3 Wave III - Runtime Models

Now that the architecture principles that accentuate the self-adaptive systems have been clarified in the second wave, we move on to concretely realize the runtime adaptation mechanism in the third wave. The article Models@run.time [6] sheds light on developing adaptation mechanisms that leverage software models for managing complexity in runtime environments. Weyns [1] aptly defines a model at runtime as: “a causally connected self-representation of the associated system that emphasizes the structure, behavior, or goals of the system from a problem space perspective.” The desire for supervising the complexity that surfaces from substantial amount of information in runtime environment led to the development of runtime models, which are at a higher level of abstraction. These models have been classified along four key dimensions, that are: Structural versus Behavioral, Procedural vs Declarative, Functional vs Non Functional and Formal vs Non-Formal. Going further, we discuss the notion of Variation Points. Appertaining to the changing conditions such as changes in context, errors, etc. in a self-adaptive system, it choses appropriate variants dynamically to comprehend variation points, changing it from one configuration to another. These variants can be more suitable in the current condition as compared to others. A safe migration path is fundamental when there is a transition between configurations. A model oriented architecture is proposed by Weyns [1] that comprehends the above approach. In this architecture, the managing system is divided into three layers, viz. Online Model Space (Top Layer), Casual Connection (Middle Layer),

and Business Application (Bottom Layer). Online Model Space is the topmost layer that is independent of the platform and only manipulates models. The other two layers are platform specific. While Casual connection, as the name suggests, has the role of linking the model space to the runtime space, where as Business Application encompasses the application logic and has sensors and factories, that track runtime events and instantiate new components, respectively. In conclusion, the third wave particularly highlights the importance of runtime models in a self-adaptive system.

2.4 Wave IV - Goal-Driven Adaptation

The knowledge of how to design managing systems is well established by now. However, researchers have not yet explicitly specified the requirements that the managing systems solve. Therefore, this wave focuses on understanding the requirements for the managing systems, in particular feedback loops. A language based approach, RELAX [7], is introduced to specify the requirements. It includes various constructs, e.g., AS CLOSE AS POSSIBLE, AS FEW AS POSSIBLE, etc., that can be used to explicitly define the constraints on the requirements. Similarly, a goal oriented approach, FLAGS [8], is presented that models the requirements by separating the goals in to crisp goals (achievement is based on boolean), and fuzzy goals (achievement is based on fuzzy constraints). These pioneer approaches enable mitigating uncertainties at runtime. This wave has also one another side because some researchers have different opinions on the requirements for the managing systems. They argue that the requirements for the managing system are the requirements about the managed system. Therefore, the design of the managing system is based on the requirements of the managed system. They call these requirements as awareness requirements [9] that can be monitored at runtime in order to analyze the runtime behaviour of the system.

2.5 Wave V - Guarantees Under Uncertainties

As it was seen in the fourth wave, uncertainty is a concern that plays a fundamental role in a self-adaptive system. Esfahni et al. in their paper propose an approach that aims to handle uncertainties in an adaptive system. It was aptly named as Taming Uncertainty in self-adaptive software [10]. Similar approaches have been discussed in this wave that emphasizes taming uncertainties; in other words, it tries to provide guarantees for the compliance of the adaptation goals. Weyns [1] argues that a software system themselves should be capable of change during runtime, as it is very ambitious to predict the changes that can be caused because of the surrounding environment or user behaviour while the application is in the development phase. The sources of uncertainties have been primarily classified into four self-explanatory parts, viz. Uncertainty related to system itself, uncertainty related to system goals, uncertainty in the execution context and uncertainty related to human aspects. Going further, a big question lies as to how can guarantees be provided for the goals, for a system that is vulnerable to uncertainties. One such approach is the Runtime Quantitative Verification (RQV) [11]. It is a mathematical approach for systems displaying stochastic behaviour that employs quantitative verification at runtime. One architecture of this approach called Quality of Service Management and Optimisation of Service-based systems, or QoSMOS [11]. Here the managed system is in the form of a Service Based System, and the managing system is termed as Autonomic Manager consisting a classic MAPE loop that uses sensors and effectors to communicate with the managed system. Going further, ActivFORMS (Active FORmal Models for Self-adaptation) [12] is another approach to provide guarantees via design-time modelling

and verification of the managing system. It adheres to the three layer model discussed in the second wave (Kramer and Magee [4]) that realises the adaptation at runtime by executing the verified MAPE loop inside a virtual machine. Moreover, using Goal Management interface, ActivFORMS supports dynamic changes of the running models.

2.6 Wave VI - Control-Based Adaptation

Engineering self-adaptive systems that provide guarantees to achieve the adaptation goals under various runtime uncertainties is challenging. Therefore, this wave focuses to design self-adaptive systems by applying the principles of control theory. Control theory is a subfield of mathematics that provides a systematic approach to design and analyze the system. It decomposes the system in to a target system (managed system), and a controller (managing system). Push-Button Methodology (PBM) [13] is a pioneer approach used in this wave to design self-adaptive systems. It works in two phases. First, model buiding phase in which the system on-the-fly runs a number of experiments on the software in order to create its linear model. Second, Proportinal-Integral-Derivative (PID) controller, a commonly used controller in self-adaptive systems, updates the parameters of the linear model to adapt the software autonomously. PBM provides various benefits such as analytical guarantees for stability, robustness, etc., hence successfully applied on a variety of self-adaptive systems [14, 15].

3 Applying Theory of Six Waves on EUREMA

In this section, we apply the theory of six waves on EUREMA. We start with a brief overview of EUREMA. Then, we present the key aspects mentioned by EUREMA under appropriate wave.

3.1 Overview

EUREMA is a model-driven engineering approach that offers a domain-specific modelling language for creating runtime models of feedback loops, and a runtime interpreter for executing the created models. At runtime, it explicitly handles the interaction between the models, the adjustment in the models, and the adaptation activities that models are responsible for. In result, the authors argue that EUREMA makes the developement of managing systems, in particular feedback loops, easier.

3.2 Wave I - Automating Tasks

EUREMA emphasizes the fact that feedback loop is an essential element in constructing a managing system in the architecture-based adaptation. Therefore, EUREMA provides support to design the managing system by explicitly modelling feedback loops and their execution. EUREMA present two types of diagrams named as Behavioral Feedback Loop Diagram (FLD), and Structural Layer Diagram (LD) to model the feedback loops. While FLD encloses the particulars of complete or partial feedback loop, LD accounts for the way the managed system and managing system are associated with each other in a self-adaptive system. This is in pursuant with the idea of MAPE-K reference model, which is a part of the managing system in wave I. Furthermore, EUREMA has been applied on different types of adaptation such as self-optimization, self-repair, etc., that also relates to wave I.

3.3 Wave II - Architecture-Based Adaptation

Self-adaptive systems can contain multiple feedback loops that operate on top of each other, i.e., layers of feedback loops. Recall that three-layer architecture model enables a higher layer to adapt lower layers, if reflection models (models that reflect the managed system and its environment) of lower layer are provided at runtime. EUREMA provides LD that allows the modelling of feedback loops and reflection models. LD presents an architectural view in which the feedback loops and reflect models are encapsulated in modules and treated as black box. Thus, EUREMA gives advantage to layer architectures. Hence, it relates to wave that focuses on the architecture principles for self-adaptive systems.

3.4 Wave III - Runtime Models

As the wave three comprehensively deliberates the runtime adaptation mechanism, it is imperative to discuss the triggering conditions for feedback loops provided by EUREMA. Triggering condition, as the name suggests, fires the execution of the feedback loop at runtime when the condition is satisfied. These triggers are nothing but events in EUREMA, either from feedback loops or the adaptable software and are only considered from those events that are emitted from the modules sensed by the feedback loop instance. This is termed, by the Authors, as Sense Relationships, that displays event flow between modules. This triggering condition is represented in EUREMA in a very simple manner, using three parameters, viz. events, period, and initialState. Events consists of list of events to be modeled; period describes the time between two consecutive executions of feedback loops, and intialState merely refers to the initial state when the executions starts. Going further, let us discuss the concept of variability modelling. Adaptation activities in the megamodel can be encapsulated and modelled, and thus, their relationships can reveal certain variation points. These variation points can be made public in the LD and can be exploited to switch between modules dynamically at runtime. This directly corresponds to the idea of variation points discussed in wave III. The authors in their context have also defined knowledge as a set of all the runtime models used by any adaptation activity. These runtime models have also been categorized in five groups, namely Reflection Models, that represent and also update with the software and its environment, Monitoring Models where the system level observations and abstraction level of reflection models are mapped. Further, the adaptation needs are identified using Evaluation Models, after which, a plan is fabricated recommending the adaptation on the reflection models which is specified by the Change Models that also defines the variability space. Finally, the Execution Model, as the name suggests, executes the planned adaptation. It should also be noted that the Evaluation Model is also used to find the appropriate adaptation using the Change Model, and therefore, Evaluation and Change models need not necessarily be different entities, and thus the combination of these models have been termed as Adaptation Models by the authors in their separate research [16]. In similar lines, the monitor and execution models together have been termed as Casual Connection models as they deal with synchronization of the software and the reflection models. This loosely follows along the lines of the model-oriented architecture for self-adaptive systems discussed in wave III, where the middle layer, also termed as casual connection encompasses components that synchronise model space to runtime space, and the top layer, called as online model space consists of components that manipulate models, in line with the adaptation models discussed above.

3.5 Wave IV - Goal-Driven Adaptation

EUREMA provides FLD to model a feedback loop including its adaptation activities known as monitor, analyze, plan, and execution. In addition, FLD enables modelling the coordination among multiple loops. On the other side, LD allows specifying the triggering conditions of a feedback loop, e.g., when a feedback loop should be executed/terminated. In nutshell, EUREMA explicitly allows to model and execute the requirements for the feedback loop. Hence, it relates to wave IV that focuses on explicitly specify the requirements for the feedback loop.

3.6 Wave V - Guarantees Under Uncertainties

EUREMA proposes the connection with the managing system and managed system using sensors and effectors, which is in conformity with QoSMOS, which is a Runtime Quantitative Verification (RQV) approach where the autonomic manager is connected with the service-based system using sensors and effectors. Similarly, in EUREMA, the casual connection between the reflection model and managed system is sustained by executing activities that use sensors and effectors via the monitor and execute activities. To handle uncertainties, EUREMA provides runtime interpreter to verify the behaviour of models, which is relevant to ActivFORMS that also executes the models in its virtual machine.

3.7 Wave VI - Control-Based Adaptation

EUREMA enables modelling parameter and structural adaptation of self-adaptive systems. Adjusting the variables in the managed system is called parameter adaptation. Whereas, structural adaptation adjusts the architecture of the managed system. Autonomic computing [17, 18] is one of the examples that applies control theory, i.e., control-based adaptation, to realize parameter adaptation. Hence, it relates to wave VI that focuses on applying control theory to relaize adaptation in self-adaptive systems. However, control-based adaptation cannot handle structural adaptation because the controller implements a specific algorithom that can only adapt the respective target system, i.e., managed system.

4 Summary

In the field of self-adaptation, it is imperative for a modelling language to flawlessly integrate design time modelling that is human driven with runtime artifacts that are machine driven. EUREMA is one such approach that comprehensively supports the explicit design of feedback loops. It covers many key aspects that relates to all the six waves. With respect to the first wave, EUREMA emphasizes the importance of feedback loops and introduces fundamental diagrams such as the Behavioral Feedback Loop Diagram (FLD) and Structural Layer Diagram (LD). Further, the concept of multiple feedback loops that can operate on top of each other is introduced which is in line with the second wave. Ideas such as Triggering Conditions and Variation Points are then put into place that is in conformity with the third wave which deals with runtime models. Moving on, smaller notions within FLD, LD are discussed that is consistent with the fourth wave and connecting intermediaries such as sensors and effectors comply with what was used in the QoSMOS architecture in the fifth wave. Finally, we discussed autonomic computing that exemplifies control theory, or control based adaptation which is consistent with the sixth wave discussed by Weyns [1].

References

[1] D. Weyns, "Software engineering of self-adaptive systems: an organised tour and future challenges," *Chapter in Handbook of Software Engineering*, 2017.

[2] T. Vogel and H. Giese, "Model-driven engineering of self-adaptive software with eurema," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 8, no. 4, p. 18, 2014.

[3] "An Architectural Blueprint for Autonomic Computing," IBM, Tech. Rep., Jun 2005.

[4] J. Kramer and J. Magee, "Self-managed systems: an architectural challenge," in *2007 Future of Software Engineering*. IEEE Computer Society, 2007, pp. 259–268.

[5] D. Weyns, S. Malek, and J. Andersson, "Forms: Unifying reference model for formal specification of distributed self-adaptive systems," *TAAS*, vol. 7, pp. 8:1–8:61, 2012.

[6] G. Blair, N. Bencomo, and R. B. France, "Models@ run.time," *Computer*, vol. 42, no. 10, pp. 22–27, oct 2009.

[7] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, and J.-M. Bruel, "Relax: Incorporating uncertainty into the specification of self-adaptive systems," *2009 17th IEEE International Requirements Engineering Conference*, pp. 79–88, 2009.

[8] L. Baresi and C. Ghezzi, "The disappearing boundary between development-time and run-time," in *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, ser. FoSER '10. New York, NY, USA: ACM, 2010, pp. 17–22.

[9] V. E. Silva Souza, A. Lapouchnian, W. N. Robinson, and J. Mylopoulos, "Awareness requirements for adaptive systems," in *Proceedings of the 6th international symposium on Software engineering for adaptive and self-managing systems*. ACM, 2011, pp. 60–69.

[10] N. Esfahani, E. Kouroshfar, and S. Malek, "Taming uncertainty in self-adaptive software," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering - SIGSOFT/FSE '11*. New York, New York, USA: ACM Press, 2011, p. 234.

[11] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic QoS Management and Optimization in Service-Based Systems," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 387–409, may 2011.

[12] M. U. Iftikhar and D. Weyns, "ActivFORMS: active formal models for self-adaptation," in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems - SEAMS 2014*. New York, New York, USA: ACM Press, 2014, pp. 125–134.

[13] 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*. ACM, 2014, pp. 299–310.

[14] S. Shevtsov and D. Weyns, "Keep it simplex: Satisfying multiple goals with guarantees in control-based self-adaptive systems," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2016, pp. 229–241.

[15] 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*. ACM, 2015, pp. 13–24.

[16] T. Vogel and H. Giese, "Requirements and Assessment of Languages and Frameworks for Adaptation Models." Springer, Berlin, Heidelberg, oct 2012, pp. 167–182.

[17] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. USA: John Wiley & Sons, Inc., 2004.

[18] K. Baclawski, M. M. Kokar, and Y. A. Eracar, "Control theory-based foundations of self-controlling software," *IEEE Intelligent Systems*, vol. 14, pp. 37–45, 05 1999.