2016 IEEE/ACM 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems

Learning and Evolution in Dynamic Software Product Lines

Amir Molzam Sharifloo[†], Andreas Metzger[†], Clément Quinton[*]
Luciano Baresi[*], and Klaus Pohl[†]
[†]paluno (The Ruhr Institute for Software Technology), University of Duisburg Essen, Germany
{amir.molzamsharifloo | andreas.metzger | klaus.pohl}@paluno.uni-due.de
[*]Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria, Milano, Italy
{luciano.baresi | clement.quinton}@polimi.it

ABSTRACT

A Dynamic Software Product Line (DSPL) aims at managing run-time adaptations of a software system. It is built on the assumption that context changes that require these adaptations at run-time can be anticipated at design-time. Therefore, the set of adaptation rules and the space of configurations in a DSPL are predefined and fixed at design-time. Yet, for large-scale and highly distributed systems, anticipating all relevant context changes during design-time is often not possible due to the uncertainty of how the context may change. Such design-time uncertainty therefore may mean that a DSPL lacks adaptation rules or configurations to properly reconfigure itself at run-time. We propose an adaptive system model to cope with design-time uncertainty in DSPLs. This model combines learning of adaptation rules with evolution of the DSPL configuration space. It takes particular account of the mutual dependencies between evolution and learning, such as using feedback from unsuccessful learning to trigger evolution. We describe concrete steps for learning and evolution to show how such feedback can be exploited. We illustrate the use of such a model with a running example from the cloud computing domain.

CCS Concepts

•Software and its engineering → Software product lines; •Computing methodologies → Machine learning;

Keywords

Adaptation, evolution, machine learning, dynamic software product lines

1. INTRODUCTION AND MOTIVATION

Adaptive systems modify themselves at run-time to react to context changes that – without adaptation – would lead to requirements violations. A well-known approach to realizing run-time adaptation is by means of a *Dynamic Software Product Line* (DSPL). A DSPL extends existing software product line engineering approaches by moving their capabilities to run-time [13]. In particular, variability binding is postponed to run-time, allowing a DSPL to activate or deactivate certain features according to context changes. Run-time reconfigurations are driven by the DSPL variability model, usually expressed as a feature model [12]. The DSPL feature model describes the possible and allowed feature combinations of the software system and thereby defines its *configuration space*.

How the system will reconfigure in response to context changes is defined by means of adaptation policies or adaptation rules [6]. To this end, the DSPL feature model is usually augmented with a set of rules that explicitly define under which circumstances a reconfiguration should take place. Adaptation rules define the actual configurations that may be reached through run-time adaptation and thereby delimit a system's *adaptation space*.

1.1 Uncertainty

DSPLs are built on the assumption that potential, relevant context changes can be anticipated at design-time. As a result, the set of adaptation rules and the possible configurations of a DSPL are predefined and fixed at design-time. Yet, for large-scale and highly distributed systems, such as cloud or cyber-physical systems, anticipating all relevant context situations and defining appropriate adaptation policies during design-time is often not possible due to the uncertainty of how the context may change at run-time [21, 9, 25].

Adding to this is the difficulty, in general, for a developer to exhaustively explore, anticipate, or resolve all possible context conditions [25]. Selecting a set of features and dependencies to be included in a DSPL involves difficult design decisions, requiring a good knowledge of the existing (or to be developed) features and the ability to predict their behavior in future contexts [10]. Similarly, developing an effective set of adaptation rules is a challenging task for developers due to the complexity and dynamicity of the context [7].

1.2 Learning and Evolution

Recently, two research directions have emerged that aim at tackling uncertainty in adaptive systems. On the one hand, *machine learning* has been proposed for modifying the adaptation space at run-time [9, 16, 14]. Context changes not anticipated during design-time are addressed by learning new adaptation rules dynamically, or by modifying and improving existing rules. On the other hand, *evolution* [4] has been proposed as a way to modify the DSPL configu-

SEAMS'16, May 16-17, 2016, Austin, TX, USA
 ISBN 978-1-4503-4187-5/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2897053.2897058

ration space [3, 24, 33]. The variability of the DSPL (in terms of features and their constraints) is changed based on insights gathered during system operation, while derived configurations are running.

1.3 Problem Statement

Learning and evolution individually address the problem of uncertainty. However, the mutual dependencies between learning and evolution also have to be considered, as context changes that cannot be addressed by learning and evolution in isolation can occur. As an example, let us assume that learning has explored the whole configuration space but none of the configurations was able to meet the requirements given the new, unforeseen context situation. In such a case, an evolution of the configuration space is required to enrich the DSPL with new features or feature combinations.

Existing adaptive system models do not explicitly consider the dependencies between learning and evolution. Well-known adaptive system models, such as the MAPE model and Kramer and Magee's reference architecture for self-management [17], are not connected with evolution, while dual life-cycle models that consider evolution, such as [22] and [20], do not explicitly reflect the role of learning and its dependency with evolution.

1.4 Paper Contributions

This paper makes three main contributions. First, we introduce an adaptive system model for dealing with uncertainty in DSPLs. Our model extends the MAPE model by learning and evolution. In particular, the model makes explicit the dependencies and feedback loops between system execution, adaptation, learning, and evolution. Second, we describe the key concepts and steps that learning and evolution have to perform as part of this model in order to exploit feedback information between learning and evolution. Third, we use an example from cloud computing to illustrate the use of the model.

The remainder of the paper is organized as follows. Section 2 describes the running example. Section 3 introduces our adaptive system model. Section 4 defines how adaptation rules can be optimized through learning, while Section 5 explains the feedback-driven evolution of DSPLs. Section 6 discusses related work. Section 7 sketches remaining challenges and concludes the paper.

2. RUNNING DSPL EXAMPLE

In a classical software product line, variability describes different possible products, *i.e.*, software systems. In contrast, DSPL variability describes different possible adaptations of the same system. Due to their foundation in software product lines, DSPLs can rely on proven engineering foundations [13]. The running system managed by the DSPL can be reconfigured by switching among configurations that are expressed in terms of a product line variability model, usually a feature model [12, 21]. A feature model serves as a compact representation of all valid feature combinations.

Figure 1 depicts the feature model of our running example, a *Software-as-a-Service* (SaaS) system that can be deployed on two types of virtual machines (`Small` and `Medium` VM) offered by a cloud provider. The XOR constraint in the feature model defines the mutual exclusion between features `Small` and `Medium`. Valid configurations of the example system are thus `C1` = {`DB, Small`} and `C2` = {`DB, Medium`}.

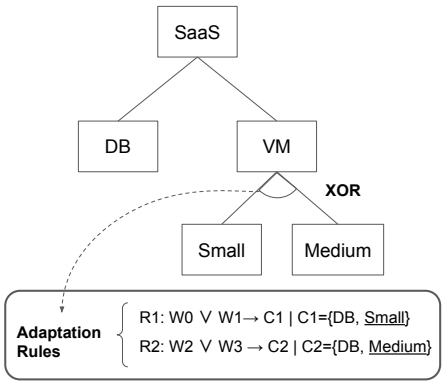

Figure 1: Feature Model (top) and Adaptation Rules

The DSPL adaptation rules relate context situations, which define the triggers for an adaptation, to feature configurations, which define the outcome of an adaptation [1, 18]. Performance (*e.g.*, here, response time) and cost are two main system requirements in our cloud example. The overall goal is to keep performance high while minimizing the cost of renting virtual machines from the cloud provider. To address these goals, the `Rule Set` in our example contains two adaptation rules `R1` and `R2` (see Figure 1). Rule `R2` switches to the `Medium` VM if the workload is high (`W2` or `W3`) to achieve performance, while rule `R1` switches to the `Small` VM if the workload is considered normal (`W0` or `W1`) to reduce costs.[1]

In our example, different kinds of uncertainty can be faced. For instance, during operation, heavier workloads way beyond what has been anticipated during design-time may occur (such as `W4`, `W5`, ...). Or, adaptation rules defined at design-time (such as choosing `C2` in case of `W3`) may be based on wrong or incomplete design-time knowledge and thus will not be effective when applied.

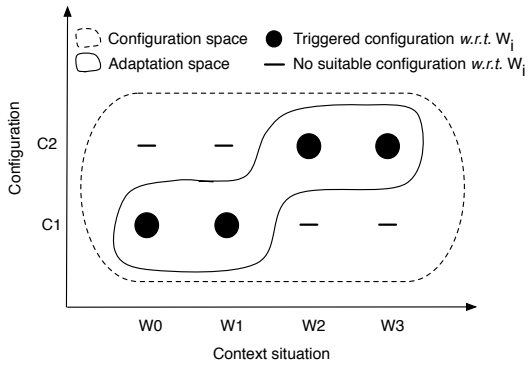

Figure 2: Configuration & adaptation spaces at design-time

In the remainder of the paper we discuss how such uncertainty can be addressed by combining learning and evolution. To support this discussion, we use a simplified, diagrammatic representation of the configuration and adaptation spaces, which is shown in Figure 2 for our example. Horizontally, the triggering conditions for an adaptation are

[1]We use a discretization of the context situations for the sake of simplicity.

shown (the workloads W0, ..., W3), while vertically the DSPL configurations are depicted (C1, and C2). The area enclosed by the dashed line depicts the configuration space, while the area enclosed by the solid line depicts the adaptation space. Black dots represent the existence of adaptation rules that are triggered by the respective context situation and adapt the system to the respective configuration. Dashes mean that a combination of context situation and configuration is not covered by any adaptation rule.

3. ADAPTIVE SYSTEM MODEL

The key elements of our adaptive system model are sketched in Figure 3. As novelties, the model adds both Learning and Evolution to the classical elements of the MAPE model (the Managed Element and the Adaptation Manager), while taking particular attention to the mutual dependencies between learning and evolution by introducing explicit feedback loops between the Adaptation Manager, Evolution and Learning.

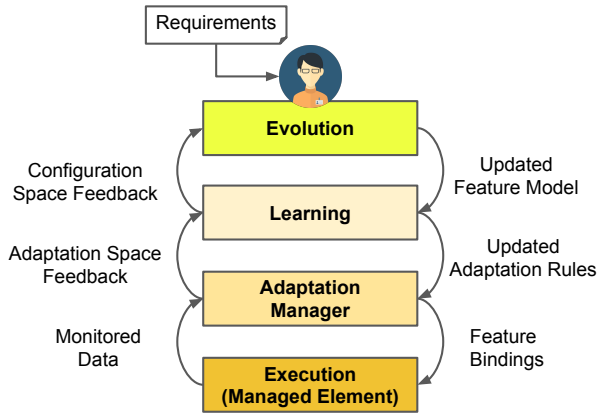

Figure 3: Learning and Evolution in DSPLs

Learning observes the execution of adaptation actions and their effectiveness in achieving requirements (Adaptation Space Feedback in Figure 3). In addition, Learning is informed by Evolution about changes in features and their constraints (Updated Feature Model) and thus can explore new configurations as part of the DSPL's modified configuration space. Based on both kinds of information, adaptation rules are revisited and updated, thus dynamically modifying the adaptation space. Updated Adaptation Rules are then fed back to the adaptation engine.

In addition to new requirements that possibly require an evolution of the DSPL (see top of Figure 3), Evolution receives from Learning insights concerning the effectiveness of different features and their combinations (Configuration Space Feedback), which constitutes further the triggers for DSPL evolution. The DSPL can be evolved by adding new features, changing feature constraints, or even by removing never used features from the DSPL feature model. While Learning is fully automatic, Evolution requires human involvement.

4. LEARNING

Learning aims at improving existing adaptation rules and adding new adaptation rules to address uncertainty faced at run-time. The learning strategy we propose in our model has been inspired by *Reinforcement Learning* (RL) [29]. In RL an agent learns the effectiveness of its actions through interactions with its context[2]. The agent receives a reward value as the feedback for applying an action. The overall aim of RL is to maximize some form of cumulative reward. Unlike supervised learning, which learns from a set of existing training data, RL relies only on the feedback from the context. This is a key advantage when handling unpredictable and changing context situations.

Given a context situation, an adaptation action is chosen by RL using two complementary strategies: *exploitation* and *exploration*. Exploitation focuses on the collected reward values, and recommends the action with the highest reward to be executed. Exploration recommends a random action regardless of its reward value to prevent the learning from sub-optimal solutions. A proper learning strategy is a reasonable combination of exploitation and exploration.

Consequently, Learning involves three major steps, which are described below: 1) adaptation rule exploitation, 2) configuration space exploration, and 3) feeding back the coverage of the configuration space to Evolution.

4.1 Adaptation Rule Exploitation

Learning starts with exploitation to collect measured data and calculate the requirements metrics (*e.g.*, average response time) to evaluate the adaptation space. Learning continuously observes the execution of adaptation rules and evaluates their ability to meet the requirements. To this end, Learning monitors a set of metrics that indicate the satisfaction of requirements. The monitoring results thereby constitute the rewards for RL. The monitoring results are stored in a data structure – we call this the Reward Set. The Reward Set contains the actual values of metrics together with the respective configuration and context situation.

In our example, the performance goal may be evaluated by measuring the average response time of the cloud service. Relating a monitored metric to the configuration of the DSPL that was active during monitoring gives the ability to reason about the effectiveness of different DSPL configurations. As a result, adaptation rules can be evaluated since they dictate which configurations are going to be used in a given context situation. In our example, the Reward Set contains the tuples $\{W0, C1, 250ms\}$, $\{W1, C1, 1000ms\}$ and so forth. Assuming our response time requirement is $500ms$, the effectiveness of the adaptation rules is evaluated and leads to the result as shown in Figure 4.

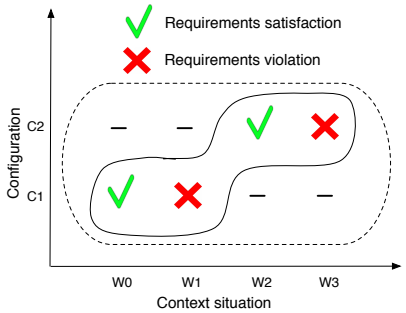

Figure 4: Adaptation Space after Exploitation

Requirements are violated in two context situations: W1

[2]aka. environment in RL.

and W3. Adaptation rules that led to requirements violations are removed and thus alternative adaptation rules can be explored in the second step.

4.2 Configuration Space Exploration

Exploration aims at improving and enhancing the set of adaptation rules. The exploration strategy examines the parts of the configuration space not yet covered by successful adaptation rules and derives new adaptation rules for these parts. As a result, configurations not yet executed in certain context situations may be encoded as adaptation rules. Figure 5 shows such optimized adaptation space after exploration in our cloud example.

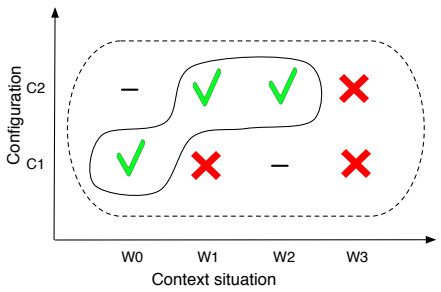

Figure 5: Configuration space after exploration

Compared to the prescribed rules, exploration has found an adaptation rule to successfully handle context situation W1 using configuration C2, instead of C1. Compared to the initial adaptation rules (see Section 2), the Rule Set is updated as follows:

$$R = \begin{cases} R1 : W0 \rightarrow C1 \mid C1 = \{DB, Small\} \\ R2 : W1 \vee W2 \rightarrow C2 \mid C2 = \{DB, Medium\} \end{cases} \quad (1)$$

In addition to improving the adaptation rules for known context situations, exploration enables us to deal with unforeseen context situations. As mentioned in Section 2, our cloud service may face a workload way beyond what has been anticipated during design-time – let it be W4. As illustrated in Figure 6, the existing adaptation rules do not support this context situation. Exploration, however, will search the configuration space with the aim of finding a configuration that satisfies the requirements for this unforeseen context situation.

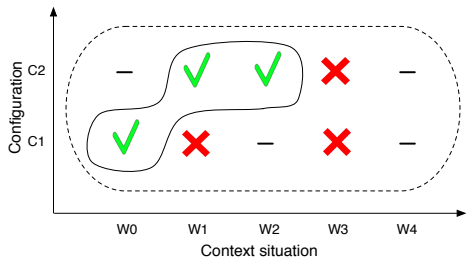

Figure 6: Unforeseen context situation

4.3 Feedback to Evolution

The exploration of the adaptation space is eventually limited to the set of features defined within the feature model of the DSPL. That is, the adaptation space may not become larger than the configuration space. In situations where none of the existing configurations is able to achieve the system goals, *e.g.*, W3 and W4 in our example, the DSPL needs to be evolved to offer more suitable features and configurations. Such need for evolution is fed back to Evolution, together with the Reward Set in order to be able to perform an analysis of the evolution need.

5. EVOLUTION

Evolution aims at updating the DSPL and its feature model to address unsatisfied or emerging requirements. Traditionally, evolution is triggered by changes in requirements that demand new features. In our adaptive system model, additional needs for evolution come from Learning by providing insights about run-time execution and in particular about the effectiveness of existing DSPL configurations. Relying on the run-time information and the expertise of developers, Evolution is to revise the design of the DSPL and its feature model towards addressing those needs. To do so, Evolution involves three major steps described below: 1) analyzing the evolution needs in terms of DSPL features, 2) updating the DSPL feature model, 3) feeding back feature model updates to Learning.

5.1 Feature-aware Analysis

To effectively address the needs for evolution, it is important to diagnose which features may be missing or misbehaving. Key information for performing such task is contained within the Reward Set. However, the Reward Set encodes the information at the level of each individual configuration. As a result, the Reward Set can be very large, depending on the size of the configuration and adaptation spaces. This makes it difficult to be used as the direct source of information for the developer who has to define the DSPL evolution.

To support the developers in decision making, we thus need artifacts that allow developers to make a decision on the feature level instead of the level of individual configurations. Here, one can rely on techniques such as statistical methods to derive "*feature-aware models*" from large sets of metric data. As an example, association rules [35] can be derived to identify the relations among features, situations, and requirements violations. Or, mathematical functions can be calculated by regression techniques to represent a system requirement as a function of DSPL feature bindings, such as the performance influence models introduced in [28].

In our example, a feature-aware model can take the following shape to understand the impact of feature selection on response time, rt, given a workload of W:

$$rt = 10 \cdot W \cdot Small + 3 \cdot W \cdot Medium \quad (2)$$

The values of $Small$ and $Medium$ are determined to be 0 or 1 depending on the activation of their corresponding features.

5.2 Feature Model Update

Considering the feature-aware models and the expected system requirements, developers re-visit the DSPL design. The current design may require further development of new

features, optimizing the existing ones, or removing ineffective features. The implementation is thus updated accordingly and the new feature model is sent to `Learning`. Additionally, developer-defined adaption rules may be added to the feature model to guide the adaptation actions and adjust the adaptation space accordingly.

Regarding our running example, the need for evolution is triggered after discovering that there exists no configuration to cope with the workloads `W3` and `W4`. The feature-aware model derived in the previous step shows the positive impact of VM capacity on the response time. This indicates that a potential solution to address `W3` and `W4` may be to increase VM capacity further. Developers then decide on the possible evolution scenarios. For instance, adding a `Large` virtual machine feature as a child feature of `VM` could be one solution. Another solution could be to add a load balancer (`LB`) and use both `Small` and `Medium` VMs simultaneously when `W3` and `W4` occur. Figure 7 depicts the latter solution in the DSPL feature model. Note that the relationship among `VM` and its children is changed to `OR` to allow the use of multiple machines in configuration `C3`. As part of the DSPL evolution, also a new adaptation rule is added that is triggered by the occurrence of `W3` and `W4`.

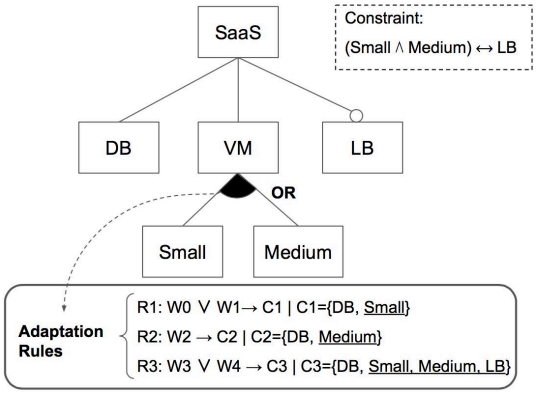

Figure 7: Evolution by Adding Load Balancer

5.3 Feedback to Learning

The evolution of a DSPL is propagated through the layers of our adaptive system model (Figure 3). In particular, a DSPL evolution may affect `Learning`. Figure 8 illustrates the changes to the configuration and adaptation space in our example. The configuration space is enlarged by the new configuration `C3` = {`DB, Small, Medium, LB`}, which reflects the addition of the load balancing feature. The adaptation space is expanded accordingly by allowing `C3` to be executed under situations `W3` and `W4`.

As shown in the figure, the accumulated information in `Reward Set` is still valid, since the previous configurations are not changed through the evolution in our example. However, for the newly introduced configuration and adaptation rule, no metric information has been collected yet and thus it is shown as bullet in the figure.

In case a feature were modified during evolution, the metrics of all the configurations that include the feature would be invalidated and thus could not be reused. In case a feature were removed, all configurations involving this feature

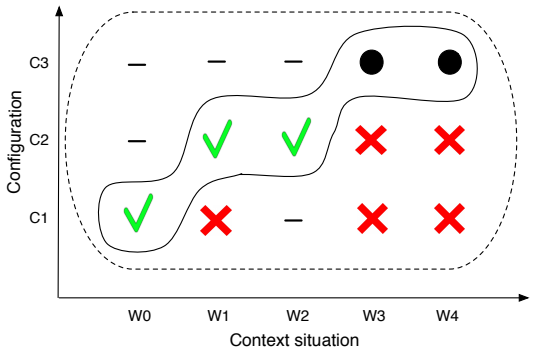

Figure 8: Evolved configuration and adaptation spaces

are removed from the configuration space. Finally, configurations that violate changed constraints among features also are to be removed. Those changes are reflected by removing the entries from the `Reward Set` accordingly.

6. RELATED WORK

This section provides a summary of the related work on learning and evolution of adaptive systems.

Existing work in learning for adaptive systems focuses on improving the adaptation decisions by analyzing the feedback from past experiences. Qian *et al.* apply case-based reasoning and store the encountered situations together with the performed adaptations [23]. When facing a new situation, similar cases are retrieved from storage to find an adaptation whose effectiveness has been shown earlier. Esfahani *et al.* discuss the application of black-box learning, in particular regression models, to understand the impact of different features on the goals of an adaptive system [9]. Features are then enabled or disabled accordingly to adapt to the changes, based on how successfully they achieve the goals. Q-Learning [32], as a specific form of reinforcement learning, has been employed to predict the impact of adaptation actions in a wide range of application domains, including data centers [5, 34, 30] and web-based systems [2]. In our recent work [14], we applied Fuzzy Q-Learning to derive explicit adaptation rules. In contrast to this previous efforts, we take a further step and propose an adaptive system model that combines learning and evolution for DSPLs.

Software evolution is a key research field in software engineering [19], and in adaptive systems it is considered a crucial concern [8]. Different techniques have been proposed to deal with run-time evolution of adaptive systems. Khakpour *et al.* [15] present a formal model for developing and modeling self-adaptive evolving systems. They rely on this model, based on evolution policies, to adapt the behavior and the structure of the system at run time, *i.e.*, to adapt and evolve it. This approach relies on predefined evolution policies, and thus cannot adapt to unforeseen evolution scenarios, while our approach is able to cope with such scenarios by relying on run-time feedback, thus continuously learning and evolving the system. Other authors propose maintaining run-time models consistent with the system they describe relying on a model-driven approach providing multiple architectural models at different levels of abstraction [31]. Those run-time models are derived from a source model that can be evolved, and thus used to propagate changes to the run-time

ones. In our previous work [3, 24], we also studied the challenges related to the evolution of DSPLs. Compared to these contributions, we go further by using run-time feedback to decide how a DSPL is evolved, and take full advantage of both the knowledge produced by machine learning and the developers' expertise.

The adaptive system model presented in this paper progresses from the state of the art of adaptive systems by studying the coexistence of learning and evolution in a unified model. It also opens up further research opportunities, which are briefly discussed below.

7. CONCLUSION & PERSPECTIVES

A DSPL is applicable if future context situations are known during design-time. Yet, for large-scale and highly distributed systems, foreseeing future context conditions and defining appropriate adaptation options during design-time is often not possible. We advocate for an adaptive system model that augments the MAPE model by a combination of online machine learning and evolution. We plan in future work to implement this model and propose an automated approach supporting learning and evoluion. One major concern when introducing machine learning into the software life-cycle is that software will change itself and morph during run-time without the explicit control of software engineers. This raises questions about the resulting overall behavior of the adaptive system. Learning may lead to unsafe or unwanted system states. We foresee different, complementary research questions to be explored to address these concerns:

Defining safe operational boundaries: Developers are able to prohibit the execution of certain configurations and adaptations by adding constraints to feature models. Indeed, learning in our adaptive system model will only explore the configuration space defined by developers. Learning only triggers an evolution (and thus exploration) of new configurations via DSPL evolution, thereby keeping the human in the loop to take the final decision.

Sandboxing: The exploration of new adaption rules and observing their effectiveness may be performed decoupled from the actual system in operation. Our adaptive system model differentiates between existing adaptation rules and the ones being explored. This information may be used to execute the rules under exploration in a sandbox.

Run-time verification: Due to the dynamic changes in the adaptation and configuration spaces due to learning and evolution, verification of an adaptive system needs to be extended to run-time [27, 11]. Existing verification techniques for adaptive systems are essentially built to check adaptations defined by developers at design-time. Advances in learning techniques (making explicit what is learned) combined with more powerful verification techniques (such as verification in the cloud) might be explored to address this challenge [26].

Acknowledgments: Research leading to these results has received funding from the EU's 7th Framework Programme FP7/2007-2013 under grant agreement 610802 (CloudWave), the DFG under the Priority Programme SPP1593: Design For Future – Managed Software Evolution (iObserve), and from project EEB - Edifici A Zero Consumo Energetico In Distretti Urbani Intelligenti (Italian Technology Cluster For Smart Communities) - CTN01_00034_594053.

8. REFERENCES

[1] G. H. Alférez, V. Pelechano, R. Mazo, C. Salinesi, and D. Diaz. Dynamic adaptation of service compositions with variability models. *Journal of Systems and Software*, 91:24–47, 2014.

[2] M. Amoui, M. Salehie, S. Mirarab, and L. Tahvildari. Adaptive Action Selection in Autonomic Software Using Reinforcement Learning. In *Fourth International Conference on Autonomic and Autonomous Systems*, pages 175–181, 2008.

[3] L. Baresi and C. Quinton. Dynamically evolving the structural variability of dynamic software product lines. In *10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS*, pages 57–63, 2015.

[4] J. M. Barnes, D. Garlan, and B. Schmerl. Evolution styles: foundations and models for software architecture evolution. *Software and Systems Modeling*, 13(2):649–678, 2012.

[5] E. Barrett, E. Howley, and J. Duggan. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *Concurrency and Computation: Practice and Experience*, 25(12):1656–1674, 2013.

[6] N. Bencomo, J. Lee, and S. O. Hallsteinsen. How dynamic is your dynamic software product line? In *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)*, pages 61–68, 2010.

[7] Y. Brun, G. Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw. Engineering Self-Adaptive Systems Through Feedback Loops. In *Software Engineering for Self-Adaptive Systems*, pages 48–70. 2009.

[8] R. de Lemos et al. Software Engineering for Self-Adaptive Systems: A Second Research Roadmap. In *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *Lecture Notes in Computer Science*, pages 1–32. Springer Berlin Heidelberg, 2013.

[9] N. Esfahani, A. M. Elkhodary, and S. Malek. A learning-based framework for engineering feature-oriented self-adaptive software systems. *IEEE Trans. Software Eng.*, 39(11):1467–1493, 2013.

[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*, ESEC/FSE '11, pages 234–244, 2011.

[11] C. Ghezzi, C. Menghi, A. M. Sharifloo, and P. Spoletini. On requirements verification for model refinements. In *Requirements Engineering Conference (RE)*, pages 62–71, 2013.

[12] 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 II*, pages 191–213, 2010.

[13] M. Hinchey, S. Park, and K. Schmid. Building dynamic software product lines. *IEEE Computer*,

45(10):22–26, 2012.

[14] P. Jamshidi, A. Sharifloo, C. Pahl, A. Metzger, and G. Estrada. Self-learning cloud controllers: Fuzzy Q-learning for knowledge evolution (short paper). In *Int'l Conference on Cloud and Autonomic Computing (ICCAC)*, pages 208–211, 2015.

[15] N. Khakpour, S. Jalili, C. Talcott, M. Sirjani, and M. Mousavi. Formal Modeling of Evolving Self-adaptive Systems. *Sci. Comput. Program.*, 78(1):3–26, Nov. 2012.

[16] D. Kim and S. Park. Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software. In *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS*, pages 76–85, 2009.

[17] J. Kramer and J. Magee. Self-managed systems: an architectural challenge. In *International Conference on Software Engineering, ISCE 2007, Workshop on the Future of Software Engineering, FOSE*, pages 259–268, 2007.

[18] J. Lee, G. Kotonya, and D. Robinson. Engineering Service-Based Dynamic Software Product Lines. *IEEE Computer*, pages 49–55, 2012.

[19] T. Mens. Introduction and Roadmap: History and Challenges of Software Evolution. In *Software Evolution*, pages 1–11. Springer Berlin Heidelberg, 2008.

[20] A. Metzger and E. Di Nitto. Addressing highly dynamic changes in service-oriented systems: Towards agile evolution and adaptation. In X. Wang, N. Ali, I. Ramos, and R. Vidgen, editors, *Agile and Lean Service-Oriented Development: Foundations, Theory and Practice*, pages 33–46. IGI Global, 2012.

[21] A. Metzger and K. Pohl. Software product line engineering and variability management: Achievements and challenges. In *Proceedings of the on Future of Software Engineering*, FOSE 2014, pages 70–84, 2014.

[22] P. Oreizy, M. Gorlick, R. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum, and A. Wolf. An architecture-based approach to self-adaptive software. *Intelligent Systems and their Applications, IEEE*, 14(3):54–62, 1999.

[23] 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 *RE'14*, pages 113–122, 2014.

[24] C. Quinton, R. Rabiser, M. Vierhauser, P. Grünbacher, and L. Baresi. Evolution in dynamic software product lines: challenges and perspectives. In *Proceedings of the 19th International Conference on Software Product Line, SPLC*, pages 126–130, 2015.

[25] A. J. Ramirez, A. C. Jensen, and B. H. C. Cheng. A taxonomy of uncertainty for dynamically adaptive systems. In *7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS*, pages 99–108, 2012.

[26] A. M. Sharifloo and A. Metzger. Mcaas: Model checking in the cloud for assurances of adaptive systems. In *To be published in Software Engineering for Self-Adaptive Systems III*, 2016.

[27] A. M. Sharifloo and P. Spoletini. LOVER: light-weight formal verification of adaptive systems at run time. In *9th International Symposium Formal Aspects of Component Software, FACS*, pages 170–187, 2012.

[28] N. Siegmund, A. Grebhahn, S. Apel, and C. Kästner. Performance-influence models for highly configurable systems. In *10th Joint Meeting on Foundations of Software Engineering*, pages 284–294, 2015.

[29] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[30] G. Tesauro, N. Jong, R. Das, and M. Bennani. A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation. *International Conference on Autonomic Computing*, pages 65–73, 2006.

[31] T. Vogel and H. Giese. Adaptation and abstract runtime models. In *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 39–48, 2010.

[32] C. J. C. H. Watkins and P. Dayan. Technical note: q-learning. *Mach. Learn.*, 8(3-4), 1992.

[33] D. Weyns, B. Michalik, A. Helleboogh, and N. Boucké. An architectural approach to support online updates of software product lines. In *9th Working IEEE/IFIP Conference on Software Architecture, WICSA*, pages 204–213, 2011.

[34] C. Xu, J. Rao, and X. Bu. URL: A unified reinforcement learning approach for autonomic cloud management. *J. Parallel Distrib. Comput.*, 72(2):95–105, 2012.

[35] C. Zhang and S. Zhang. *Association Rule Mining: Models and Algorithms*. Springer-Verlag, 2002.