Dynamic re-configuration of software product lines towards an exploratory study on DSPLs

Danillo Sprovieri

Dynamic Re-Configuration of Software Product Lines

Towards an Exploratory Study on DSPLs

Danillo Sprovieri
Centre de Recherche en Informatique
Université Paris-Sorbonne
Paris, France
danillo.sprovieri@malix.univ-paris1.fr

Abstract*—Context:*** **Adaptations need to be considered at design-time (adapting complex systems to new technologies, reengineering due to new regulations etc.), but also during run-time (e.g. new emerging functional and non-functional requirement, context-specific decisions).** *Objective:* **I use SPLs as a strategy for coping with uncertainty and adapting to change, where conventionally change occurs in the requirements of the software product lines' market. Our idea is to design a variability mechanism in the domain of dynamic software product lines engineering in order to enable continuous evolution and adaptation of the software product lines at run-time.** *Method:* **I investigate dynamic change propagation of SPLs at run-time through an explorative study. A literature review and semi-structured personal interviews with relevant actors in the domain of SPLs are the fundament of our research. This analysis enables us to understand how SPLs are dynamically adapted and evolved in practice.** *Conclusion:* **This study will give us an overview of the domain of DSPLs and allows us to identify the research gap regarding run-time adaptation and evolution of SPLs.**

Keywords—Run-Time Adaptation; SPLs; DSPLs

I. INTRODUCTION

Due to market dynamics, changing business conditions and new regulations [1], the success of an enterprise depends increasingly on its ability to be flexible and react to environmental changes in a quick and cost-effective way [2]–[6]. Under these conditions, adaptations are unavoidable and crucial in most application domains [4], [6]–[10]. These adaptations need to be considered at design time (adapting complex systems to new technologies, reengineering due to new regulations, etc.) [11], [12], but also during run-time (e.g. context-specific decisions) [13], [14]. For several decades, all software was custom software. Every application was tailored specifically to one customer's requirements. Software reuse has gained more and more attraction from the research community to find a solution for developing complex systems. Reuse software parts leads to improvements such as productivity, quality and cost reduction [15]. The idea of software reuse was already introduced by McIlroy [16], who proposed at that time mass production of software components as the base for software development. Later on, Parnas [17]–[19] created the artefact of program family, which is the foundation until today in engineering for reusable component and reuse-based application development [20]. There are several advantages to the product lines production strategy. According to the study realized by Clements and Northrop [21] the product lines production approach decreases not only the cost per product, but also the time to market, the labor need and improves the productivity, the quality of each derived product and increases the portfolio size and therefore the possibility to gain new markets. One of the most effective means to manage software reuse in an industrial context is software product lines engineering. In particular, product lines engineering allows managing a family of applications that share a common architecture. These applications are configured as a set of components that provide core functions, but will exhibit variability in the features they provide to users because each family member will use a unique set of non-core components bound to the architecture at variation points.

In parallel with advances in software engineering for SPLs, a new generation of middleware and service-oriented architectures has emerged that is capable to adapt its behaviour according to changes in operation or environment conditions, and/or user requirements. This may require a system to adapt dynamically to environmental change in a way that is far more radical than is possible using mere modes of operation supported by parametric adaptation [22]. The emergence of such re-configurable software machinery has profound implications for software development. In particular, it permits: Software to be developed that can tolerate a wide range of environmental contexts without forcing the developer to enumerate how the system will behave in each; Software to be developed in the presence of uncertainty [23] about the environmental contexts that may be encountered at run time. Effectively, a re-configurable capability permits design decisions to be deferred from design time to run time. This is only possible if the system is able to monitor its environment at run time, using the monitoring data and the system's own internal state to trigger adaptations as necessary. However, such adaptations must be driven by the need to satisfy the system's requirements. However, there is currently a wide conceptual gap between the capabilities of re-configurable machinery, expressed in terms of components and services, and requirements, expressed in terms of user or customer needs, with only an emerging understanding of what changing environmental context means. The latter characterizes the extent to which a system can tolerate change in the environment; from optimizing the way a fixed set of requirements are satisfied, to adapting to satisfy new requirements that emerge dynamically.

In this paper I intend to study this aspect in the context of adaptive middleware and service-oriented architectures. The possible solutions will be empirically evaluated to improve a selected solution that will be implemented into a tool be used by software practitioners and other research laboratories around the world.

A. *Software Product Lines*

In an SPL, "a set of software-intensive systems …. share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a pre-scribed way"[1]. Thus, a family of applications will share a common architecture, configured as a set of components that provide core functions, but will exhibit variability in the features they provide to users because each family member will use a unique set of non-core components bound to the architecture at variation points.

II. Challenges and Problematic Situation

It is not possible to anticipate all possible execution contexts already at design-time (e.g. if the change is a user interface that must be adapted to an as yet unknown UI technology). At design-time, the software product lines must be specified in a way that is able to sense and react to context changes at run-time. So, the system is able to cope to a certain degree with new emerging requirements. Andersson et al. [24] characterize the scope of change as follows: (i) taken care of, (ii) planned for or (iii) not planned for. Adapting dynamically to satisfy new requirements is unplanned change and for fully autonomous systems is currently possible only in the realm of science fiction. Adaptive middleware and service-oriented architectures are now able to support change, but dealing with change that is planned for but not taken care of at design-time is an open challenge that, when solved, will significantly extend the types of problems for which resilient software systems can be developed. However, even where adaptation involves optimizing for a fixed set of requirements (as is the case for (i) and (ii)), established techniques for reasoning about requirements are insufficient, since not only must the system's behaviour be specified when the environment is in a steady state, but the adaptive behaviour (i.e. under what circumstances must the system adapt?) have to be specified also. Worse, if, as is frequently the case, knowledge about the environment is incomplete at design time, the system's behaviour must be specified not only for the environmental contexts that the analyst believes the system will encounter, but also for those the system will encounter if the analyst's belief evidences to be mistaken. Risks of evolution are mainly related on the one hand to loss of consistency and accuracy in the components and on the other hand the loss of the benefits associated with the product lines technique. These risks are increasing directly proportional to the number of components and the complexity of the relations between them. Given the importance and the need for solutions that avoid these risks [25], L. Yu and S. Ramaswamy [26], Dhungana et al. [27], Borba [28], Neves [29], have made several proposals to address these challenges. However, the arguments and empirical evidences presented by Dhungana et al. [27] and Neves [29] suggest that the few existing proposals are far from representing a satisfactory solution to the problems and challenges associated to the evolution of PLs systems. The use of SPLs can be seen as a strategy for coping with uncertainty and adapting to change, where conventionally, change occurs in the requirements of the software product lines' market. In contrast to autonomous re-configurable systems, the derivation of new SPLs variants typically involves satisfying new goals rather than the ways in which a fixed set of goals are realized. Moreover, these goals could not be included when the SPLs was conceived. Thus, SPLs deal with unplanned change. This is possible because adaptation is done offline by human developers rather than automatically at run-time. Nevertheless, the techniques used in SPLs are exploitable. Thus, a re-configurable system can be conceptualized as a dynamic SPLs (DSPLs) [30], [31]; a set of variants within a product family, with each variant optimized to a particular environmental context. The key difference between an SPLs and a DSPLs is the time at which variants are bound; at design or deployment time for SPLs or at run-time for DPSLs.

III. Related Work

Lapouchnian [32] defines Awareness Requirements (or AwReqs) as requirements that talk about the success or failure of other requirements. AwReqs can refer to goals, tasks, quality constrains and domain assumptions. However, while I plan to use in-memory representation and a limited form of manipulation of goal models, Lapouchnian does not deal with the runtime representation of the requirements, i.e. not explicit runtime representation of AwReq is provided. Instead, they focus on the mapping from requirement models to feedback loops using the requirements monitoring framework proposed by Robinson [33] to monitor AwReq. Lapouchnian [32] does reasoning about partial satisfaction of requirements. They offer high-level monitoring capabilities that can be used to determine satisfaction levels for AwReqs. In our case, the DSPL itself verifies if an assumption is true. If the assumption is falsified, the system automatically re-evaluates the trade-off among the softgoals, triggering an adaptation to rebalancing the softgoals if necessary. Kirsch-Pinheiro et al. [34] explored the re-adaptation of context-oriented systems proposing a roadmap to context management and a requirements elicitation process. DeLoach & Miller [35] explore how to maintain a runtime representation of goals. However, they do not deal with the runtime representation of softgoals or goal realization strategies. As far as I am able to understand, in their research the running system interacts with the runtime goal model to trigger an update of the runtime goal model (a goal can be triggered to go from active to achieved, failed, obviated, or removed). Its main utility has been for understanding what the system is doing in terms of goals. No reasoning about partial satisfaction is done. This contrasts with Letier & Van Lamsweerde [36], which formalizes a mean for representing partial goal satisfaction based on KAOS. A contrasting approach to partial goal satisfaction is covered by RELAX proposed by Whittle et al. [37]. Although RELAX is not goal-based per se, Cheng et al. [38] illustrates the use of RELAX, with KAOS goal models, using obstacle analysis to identify

[1] www.sei.cmu.edu/productlines

when to RELAX a goal. Baresi & Pasquale [39] propose adaptive goals that are aware of their own degree of satisfaction during runtime and a means to trigger adaptation.

Raúl Mazo et al. [40] presented a Java-based tool (VariaMos) for defining variability modeling languages, (dynamic) product lines and re-configurable systems. In addition to that, the tool enables automated verification, analysis, configuration and simulation. Models designed with VariaMos refer to the specification of variability of dynamic product lines. So, it enables at a certain degree to modify the modeling languages at run-time. In the same place, at the same time, Muñoz-Fernández et al. [41] investigated in more detail the requirements specification for re-configurable systems. They presented a multi-view framework. This framework fosters to manage uncertainty and is sufficiently expressive for re-configurable systems.

IV. Research Design

In this Section, I outline how I conduct research and the reasoning about the applied concepts.

A. Research Goal

I investigate, how the idea of dynamic software product lines could help to deal with the challenges of developing efficient re-configurable software. I also offer insight into the different approaches that use dynamic software product lines engineering for developing re-configurable systems focusing on practical approaches

B. Research Questions

The research goal, combined with the problems and approaches mentioned in Section 1 lead to the following research questions:

RQ1. What elements are common in evolutionary and re-configurable SPLs? Why is it not possible to anticipate at design-time all re-configurations and what are the requirements to adapt and evolve at run-time? The answer should help to establish a conceptual architecture to facilitate reasoning about conceptual adaption. Literature review and analysis of real interviews are fundamental to identify relevant information about how to re-configure and evolve SPLs at run-time.

RQ2. How can I support run-time re-configuration and evolution of SPLs? Is the context required to evolve from dynamic adaptation to dynamic re-configuration? The answers to these question refers to the main goal of this project. The information is required to suggest and develop a solution that can be applied to derive a solution of which configurations are appropriate to a given context at run-time. Dynamic software product lines engineering is a mean to flexibly handle context changes at run-time. Techniques in this area are used to develop an architecture that support run-time re-configuration of re-configurable systems.

RQ3. How can the conceptual architecture be validated? This should establish a validation framework.

C. Research Strategy

Based on the problem description, I chose Design Science Research (DSR), proposed by Hevner et al. [42] and V. Vaishnavi and B. Kuechler [43], as a research strategy, as my goal is to derive an artefact, which is able to provide empirical evidence for filling the research gap. Also, because DSR determines iteratively the reality and literature foundation that emerge from research efforts. DSR as research strategy provides both the methodology of design science research by Vaishnavi et al. [43], [44] and the information system research framework by Hevner and Chatterjee [45]. The methodology describes the process steps and the circulation of knowledge within these steps. The framework considers previous scientific work (rigor cycle) and a real environment (relevance cycle), during the artefact development and evaluation in the field of information systems.

To apply the research strategy, the approach is based on the methodology of design science research by Vaishnavi et al. [43] and further developed the framework of information system research by Hevner and Chatterjee [45]. Interviews with domain experts serve as our environment and the literature provide evidence for flexible adaption of re-configurable systems at run-time is a real issue to solve and therefore has the need for a framework to support re-configurable adaptions. The research methodology is illustrated and structured in five phases. In the subsequent Section, I elaborate in more detail how these phases are executed. The knowledge base in the right column refers to the literature. I take existing re-configurations methods into account. The framework considers both applicable knowledge (rigor cycle) and business needs (relevance cycle) during the artefact development and evaluation in the field of information systems. The rigor cycle refers primarily to the literature review in order to define the research problem. The literature is also referred in the suggestion and development of the artefact. I choose appropriate techniques and improve or adapt what is already available. The relevance cycle serves to integrate the problem domain into the research by deriving the needs from the interview sections and applying them to the artefact. It thus supports the alignment of the solution with the environment. The rigor and relevance cycles are applied iteratively during the execution of the five phases.

In the Section research methodology, I describe how the methodology of Vaishnavi and Kuechler [43] is applied on this research. Specifically, I outline how the phases of the methodology are related to the research questions and thus the research goal. Next, I explain how the problem domain affects the research approach. An inductive approach is conducted in order to develop a new artefact for supporting run-time adaptation and evolution of SPLs.

D. Research Methodology

In this Section, I outline the applied research methodology of Vaishnavi and Kuechler [43]. The research methodology structures, defines and justifies the research procedure applied, how I conducted research, and how results are achieved. The methodology is divided into five phases: awareness of the problem, suggestion, development, and evaluation. The

following section describes the main characteristics of the five phases of the methodology.

1) Awareness of the Problem

In the first phase awareness of the problem, I gain an in depth understanding of the problems that should be solved with the developed artefact. In order to do so, I identify the requirements of run-time evolution and adaptation of SPLs. More precisely, I identify relevant information about why it is not possible to predict all re-configurations at design-time and what are the requirements to evolve at run-time by analyzing the process and arising problems found in the literature. It represents the problem where change can be planned at design-time. Doing so, the needs are properly reflected and are sound to build a solution upon. In order to gain a broader overview of the different facets of the problem, I conduct a literature review of concepts for similar solutions. To strengthen the relevance cycle, I investigate if the process contains the same issues found as during the literature review, which led to the problem description (**RQ1**).

2) Suggestion

In the second phase, based on the insight of the previous phase, I present a concept for a possible solution. First, I analyze the previously found requirements to find a solution for the investigated scenario. This information is needed to suggest a solution that supports run-time re-configuration and evolution of SPLs. As a next iteration in the rigor cycle, the derived requirements are validated against the problem description. This proofs that the proposed solution is valid according to the problem domain. Furthermore, the suggestion is ground while comparing it to similar solutions found in the literature review. This phase describes how to develop the desired artefact in order to solve the identified problem (**RQ2**).

3) Development

In the third phase, I elaborate the suggested solutions according to the results of the requirements analysis (relevance cycle iteration). To develop the artefact, I use dynamic product lines engineering techniques and design architecture, which support run-time re-configuration and evolution of SPLs. The run-time adaptation and evolution is demonstrated on the basis of real practices. The overall goal of this phase is to design an architecture, which supports run-time adaptation and evolution of SPLs (**RQ2**).

4) Evaluation

In the fourth phase (evaluation), I perform a validation of my results. It provides empirical evidence that the needs are reflected properly by using real instances to test the functionality of the architecture. The results of the evaluation provide answer to the question, if it is possible to evolve and re-configure SPLs at run-time (**RQ3**).

5) Conclusion

Finally, in the fifth phase (conclusion), the results are critically discussed.

In addition, I conduct in each phase of the research methodology personal interviews with relevant actors in the domain of SPLs. This approach strengthens the relevance cycle and increases the quality of research. The research presented has a qualitative approach and the aim of an exploratory nature.

V. Action Plan

The main aim of the action plan is to conduct a study on re-configurable and evolutionary SPLs involving literature and prominent actors in the software engineering communities and technology industries. All participants are experienced software engineers. The results of the action plan will be reported as an empirical study conducted with software engineers from different domains (e.g. technology industries) and will be useful for further research in the software engineering community. In addition, the study will be empirically validated.

The goal of the action plan is to elicit the requirements of re-configurable and evolutionary SPLs. To achieve the goal, the action plan has been organized into several process steps:

A. Identification and Formalization of the Problem Domain

The first process step is conceived as the very first interaction point of the action plan where I define and elicit the need and the vision for run-time re-configuration and evolution of SPLs. Specifically, I discuss the requirements from research, methodology, technical, architectural and user-perspectives. Indications about constraints and standards will also be defined supporting the final architecture. This phase explicitly represents the rationale about the study and determines the research gap identified during the literature review. So, new novel theory and hypotheses can be generated.

B. Objective and Preparation

To execute the reviewing activities, we use the general process of reviewing protocols proposed by Kitchenham and Charters [46]. We refer to the reviewing process that has been conceived with a particular emphasis on reviews conducted within the domain of software engineering [46]. To systematically retrieve research works in the domain of DSPLs, I perform a search using carefully a planned search query on scientific electronic databases that were accessible online (e.g. IEEExplorer, ACM Digital Library, Citeseerx Library, ScienceDirect, SpringerLink). The query will be designed based on keywords derived from the research questions (**RQ1** – **RQ3**). This systematic literature review intends to provide an objective summary of the existing knowledge in the area of DSPLs. At the same time, I want to highlight open issues in order to suggest the areas that are in need of further contributions. Based on the gained knowledge, I will create together with domain experts a protocol for the interview sections. Important is the fact that there must be no limitation to new ideas occurring during the interview sections. So, for example questions can be added or modified during run-time. Each interview should start with general questions about the participants and their experiences. Next, I ask them which challenges they face during engineering re-configurable and evolvable SPLs. Then, possible research initiatives, which could be helpful in their current organization, are discussed. The objective is refined into three questions, which will be answered through the data collection and analysis:

- How do they deal with unplanned changes?

- Have they experienced a problem with unplanned changes?
- And if there is a problem: What do they expect from a solution?

An important element of the preparation is the implementation of a pilot test. The first interview will be conducted in our own laboratory and will support to improve the quality of the study. Possible weaknesses can be uncovered and eliminated.

C. Interview Design

I will conduct semi-structured interviews, which allow to consider new arising ideas during the interview sections. In general, open-ended questions are primarily. We specify a protocol that defines at design-time the general topics of interest. This relieves the collection of similar information from all participants. The protocol contains a list of questions and topics that need to be covered during run-time. The protocol can be modified depending on the appropriateness during the conversation. Since the data will be collected from human beings, it is very important to carefully select participants for the interview sections. This fosters a more reliable and valid study.

D. Collection, Analysis, and Formalization of Results

This process step involves collection and analysis of quantitative and qualitative data. Furthermore, it supports the formalization of the results of both research and development. This takes place in strict cooperation with domain experts. I will use multiple sources of data to create the study database. The collection of data will be stored in a semi-structured way for processing, analysis and dissemination. Resulting from this phase, criteria for interpreting the collected data should be identified and it should be also identified, which data element referrer to which research question.

E. Validation

We plan to apply an observational method (e.g. case study) to validate our architecture. This allows us to monitor and collect data over the whole study with a relatively minimal addition to cost. Valuable information can be obtained characterizing the development of the artefact. The main goal of the validation is to demonstrate a causal relationship between outcome and intervention. Furthermore, it must be analyzed in more detail, if the findings can be further generalized and applied to different domains.

F. Dissemination of Results

The last step deals with supporting activities including dissemination, exploitation, and standardization are undertaken throughout the whole action plan. Specific documentation, presentations, press releases and other materials will be prepared and disseminated to encourage further research and ongoing development of the middleware and its related technologies also after action plan completion.

VI. Conclusion

In this paper, I present the scientific rationale of the exploratory study. The results of this study will be useful to software engineering professionals and researchers worldwide by revealing fundamental concepts of re-configurable and evolving SPLs. The study could raise new research directions: (i) what are the challenging experiences, (ii) research initiatives, (iii) best practices.

A. Future Work

I will use the results of the interview sections and combine them with the findings of the literature (e.g. DSPLs techniques) in order to define a middleware that supports dynamic adaptation and evolution of SPLs. The aim is to develop an architecture of the middleware. The architecture will permit to the technical and research work to establish their interconnections. We provide feedback on the architecture of the middleware with respect to the elicited requirements. The assessment will be done to constantly check that research is done correctly with respect to the expectations. In addition, gathering empirical evidence is already included in our working agenda. Nonetheless, some further work is also required to perform run-time reasoning.

Acknowledgment

This Ph.D. study is part of research work funded by the Ministère de l'Enseignement supérieur et de la Recherche (MESR). The research project is supervised by Prof. Carine Souveyet (carine.souveyet@univ-paris1.fr), Université Paris-Sorbonne, France.

References

[1] D. Goodhue, D. Che, M. C. Boudreau, and A. R. Davis, "Addressing business agility challenges with enterprise systems," MIS Q. Exec., vol. 2, no. 8, pp. 73–88, 2009.

[2] C. Li, M. Reichert, and A. Wombacher, "Mining business process variants: Challenges, scenarios, algorithms," Data Knowl. Eng., vol. 70, no. 5, pp. 409–434, 2011.

[3] B. Mutschler, M. Reichert, and J. Bumiller, "Unleashing the Effectiveness of Process-Oriented Information Systems: Problem Analysis, Critical Success Factors, and Implications," IEEE Trans. Syst. Man, Cybern. Part C (Applications Rev., vol. 38, no. 3, pp. 280–291, 2008.

[4] R. Lenz and M. Reichert, "IT support for healthcare processes - premises, challenges, perspectives," Data Knowl. Eng., vol. 61, no. 1, pp. 39–58, 2007.

[5] T. H. Davenport, "Mission Critical: Realizing the Promise of Enterprise Systems," Feb. 2000.

[6] P. Dadam, M. Reichert, and S. Rinderle-Ma, "Prozessmanagementsysteme," Informatik-Spektrum, vol. 34, pp. 364–376, 2011.

[7] W. M. P. Van Der Aalst, "A decade of business process management conferences: Personal reflections on a developing discipline," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 7481 LNCS, pp. 1–16, 2012.

[8] F. Casati, S. Ceri, B. Pernici, and G. Pozzi, "Workflow evolution," Data Knowl. Eng., vol. 24, no. 3, pp. 211–238, 1998.

[9] S. Rinderle, M. Reichert, and P. Dadam, "Correctness criteria for dynamic changes in workflow systems - A survey," Data Knowl. Eng., vol. 50, no. 1, pp. 9–34, 2004.

[10] H. a. Reijers, S. Van Wijk, B. Mutschler, and M. Leurs, “BPM in practice: Who is doing what?,” Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 6336 LNCS, pp. 45–60, 2010.

[11] A. Hallerbach, T. Bauer, and M. Reichert, “Managing process variants in the process life cycle,” in ICEIS 2008 - Proceedings of the 10th International Conference on Enterprise Information Systems, 2008, vol. 2 ISAS, pp. 154–161.

[12] M. Rosemann and W. M. P. Van Der Aalst, “A Configurable Reference Modeling Language,” Inf. Syst., vol. 32, no. 1, pp. 1–23, 2007.

[13] B. Weber, M. Reichert, and S. Rinderle-Ma, “Change patterns and change support features - Enhancing flexibility in process-aware information systems,” Data Knowl. Eng., vol. 66, no. 3, pp. 438–466, 2008.

[14] M. Reichert and P. Dadam, “ADEPTflex - supporting dynamic changes of workflows without losing control,” J. Intell. Inf. Syst., vol. 10, no. 2, pp. 93–129, 1998.

[15] K. Pohl, G. Böckle, and F. J. van der Linden, Software Product Line Engineering: Foundations, Principles and Techniques. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[16] M. D. McIlroy, “Mass Produced Software Components,” Nato Sci. Committe Softw. Eng., 1969.

[17] D. L. Parnas, “On the criteria to be used in decomposing systems into modules,” Commun. ACM, vol. 15, no. 12, pp. 1053–1058, 1972.

[18] D. L. Parnas, “On the Design and Development of Program Families,” IEEE Trans. Softw. Eng., vol. SE-2, no. 1, pp. 1–9, 1976.

[19] D. L. Parnas, “Designing Software for Ease of Extension and Contraction,” IEEE Trans. Softw. Eng., vol. SE-5, no. 2, pp. 128–138, 1979.

[20] K. C. Kang, V. Sugumaran, and S. Park, Applied Software Product Line Engineering, 1st ed. Boston, MA, USA: Auerbach Publications, 2009.

[21] P. C. Clements and L. Northrop, “Software Product Lines: Practices and Patterns,” SEI Ser. Softw. Eng. Addison-Wesley Prof., 2001.

[22] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng, “Composing adaptive software,” IEEE Comput., vol. 37, no. 7, pp. 56–64, 2004.

[23] K. Welsh and P. Sawyer, “Understanding the scope of uncertainty in dynamically adaptive systems,” in Proc. Sixteenth International Working Conference on Requirements Engineering: Foundations of Software Quality (REFSQ’10), 2010, vol. 6182 LNCS, pp. 2–16.

[24] J. Andersson, R. de Lemos, S. Malek, and D. Weyns, “Modeling Dimensions of Self-Adaptive Software Systems,” in Software Engineering for Self-Adaptive Systems SE - 2, vol. 5525, B. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Springer Berlin Heidelberg, 2009, pp. 27–47.

[25] J. D. McGregor, “The Evolution of Product Line Assets,” IEEE Multimed., vol. 10, no. June, pp. 4–8, 2003.

[26] L. Yu and S. Ramaswamy, “A Configuration Management Model for Software Product Line,” INFOCOMP J. Comput. Sci., vol. 5, no. 4, pp. 1–8, 2006.

[27] D. Dhungana, T. Neumayer, P. Gruenbacher, and R. Rabiser, “Supporting the Evolution of Product Line Architectures with Variability Model Fragments,” Seventh Work. IEEE/IFIP Conf. Softw. Archit. (WICSA 2008), pp. 327–330, 2008.

[28] P. Borba, L. Teixeira, and R. Gheyi, “A theory of software product line refinement,” Theor. Comput. Sci., vol. 455, pp. 2–30, 2012.

[29] L. Neves, L. Teixeira, D. Sena, and P. Borba, “Investigating the Safe Evolution of Software Product Lines,” in 10th International Conference on Generative Programming and Component Engineering (GPCE 2011), 2011, no. Section 4, pp. 33–42.

[30] S. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid, “Dynamic Software Product Lines,” in Systems and Software Variability Management SE - 16, R. Capilla, J. Bosch, and K.-C. Kang, Eds. Springer Berlin Heidelberg, 2013, pp. 253–260.

[31] P. Sawyer, R. Mazo, D. Diaz, C. Salinesi, and D. Hughes, “Constraint Programming as a Means to Manage Configurations in Self-Adaptive Systems,” Spec. Issue IEEE Comput. J. “Dynamic Softw. Prod. Lines,” vol. 45, no. October 2015, pp. 56–63, 2012.

[32] A. Lapouchnian, “Exploiting Requirements Variability for Software Customization and Adaptation,” in Foundations of Software Quality (REFSQ’10), 2011.

[33] N. Robinson, “A Requirements Monitoring Framework for Enterprise Systems,” Requir. Eng., vol. 11, no. 1, pp. 17–41, 2005.

[34] M. Kirsch-pinheiro, R. Mazo, C. Souveyet, and D. Sprovieri, “Requirements Analysis for Context-oriented Systems,” 7th Int. Conf. Ambient Syst. Networks Technol. (ANT 2016), pp. 1–8, 2016.

[35] S. A. DeLoach and M. Miller, “A goal model for adaptive complex systems,” Int. J. Comput. Intell. Theory Pract., vol. 5, no. 2, 2010.

[36] E. Letier and A. Van Lamsweerde, “Reasoning about partial goal satisfaction for requirements and design engineering,” in Proc. of Twelfth ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2004.

[37] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, and J.-M. M. Bruel, “RELAX: a language to address uncertainty in self-adaptive systems requirement,” Requir. Eng., vol. 15, no. 2, pp. 177–196, 2010.

[38] B. H. C. Cheng, P. Sawyer, N. Bencomo, and J. Whittle, “A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty,” in in ACM/IEEE Twelfth Int. Conference On Model Driven Engineering Languages And Systems (MODELS 09), 2009, vol. 5795 LNCS, pp. 468–483.

[39] L. Baresi, L. Pasquale, and P. Spoletini, “Fuzzy Goals for Requirements-Driven Adaptation,” in 18th IEEE International Requirements Engineering Conference, 2010, pp. 125–134.

[40] J. C. Muñoz-fernández, C. Salinesi, R. Mazo, J. C. Muñoz-fernández, L. Rincón, C. Salinesi, and G. Tamura, “VariaMos : an Extensible Tool for Engineering (dynamic) Product Lines VariaMos : an extensible tool for engineering (dynamic) product lines,” no. September, 2015.

[41] J. C. Muñoz-Fernández, G. Tamura, M. Raúl, and C. Salinesi, “Towards a Requirements Specification Multi- View Framework for Self-Adaptive Systems,” in Computing Conference (CLEI), 2014 XL Latin American, 2014, pp. 1–12.

[42] A. R. Hevner, S. T. March, J. Park, and S. Ram, “Design Science in Information Systems Research,” MIS Q., vol. 28, no. 1, pp. 75–105, Mar. 2004.

[43] V. Vaishnavi and B. Kuechler, “Design Science Research in Information Systems,” Ais, p. 45, 2004.

[44] V. K. Vaishnavi and W. Kuechler Jr., Design Science Research Methods and Patterns: Innovating Information and Communication Technology, 1st ed. Boston, MA, USA: Auerbach Publications, 2007.

[45] A. Hevner and S. Chatterjee, Design Research in Information Systems: Theory and Practice. 2010.

[46] B. Kitchenham and S. Charters, “Guidelines for performing Systematic Literature Reviews in Software Engineering,” Engineering, vol. 2, p. 1051, 2007.