In Proceedings of the Workshop on Modularization, Composition and Generative Techniques for Product Line Engineering (McGPLE), held in conjuction with GPCE / OOPSLA 2008, Nashville, Tennessee, October 19-23 2008.

Modelling Variability in Self-Adaptive systems: Towards a Research Agenda

A. Classen[⋆∗] A. Hubaux[⋆] F. Sanen[†] E. Truyen[†] J. Vallejos[‡]
P. Costanza[‡] W. De Meuter[‡] P. Heymans[⋆] W. Joosen[†]

[⋆]PReCISE Research Centre, Faculty of Computer Science, University of Namur
{acs,ahu,phe}@info.fundp.ac.be

[†]Distrinet Research Group, Department of Computer Science, K. U. Leuven
{frans.sanen,eddy.truyen,wouter.joosen}@cs.kuleuven.be

[‡]Programming Technology Lab, Vrije Universiteit Brussel
{jvallejo,pascal.costanza,wdmeuter}@vub.ac.be

Abstract

The combination of generative programming and component engineering applied to software product line engineering (SPLE) has focused thus far mostly on static systems (as previous editions of AOPLE indicate), with variability that is bound once. Meanwhile, an emergent paradigm in software engineering deals with self-adaptive and dynamic systems. While there is a well-known and agreed SPLE process for static systems, there has been less focus on dynamically adaptive systems. As such it appears imperative to include it in an extended research agenda.

In the present paper we observe limitations related to domain engineering in SPLE and identify what fundamental concepts, such as context *and* binding time*, must be retought in order to achieve SPLE for dynamically adaptive systems. The main contribution of this paper is a set of research questions, aimed at defining a common research agenda for addressing these limitations.*

1 Introduction

As previous editions of the AOPLE workshop indicate, the combination of generative programming and component engineering applied to software product line engineering (SPLE) has focused thus far mostly on systems with static variability binding. Meanwhile, an emergent paradigm in software engineering deals with self-adaptive systems (viz. SEAMS workshop at ICSE, or DSPL at SPLC). Self-adaptive systems are systems that are able to autonomously adapt to changing circumstances without human intervention, or, in other words, systems that are able to cope with dynamic variability. To support dynamic variability, a self-adaptive system typically implements a control loop that consists of monitoring the application and its context, analyzing the situation, planning and executing any required adaptations (as depicted in Figure 1).

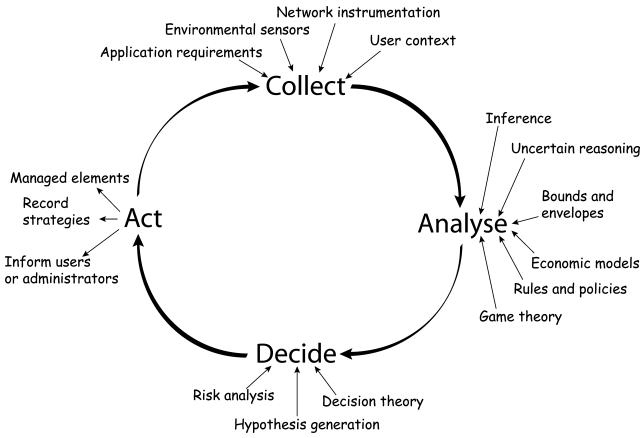

Figure 1. Typical control loop (from [6])

System-specific adaptation knowledge is used to specify when, where and how to adapt the system. This adaptation knowledge is typically specified by developers as adaptation rules following the well-known Event-Condition-Action format. For example, the Rainbow framework from Garlan et al. [8] allows to express rules like "when the response time observed by a client exceeds a well-defined threshold, and the server is overloaded, then migrate the client to a less loaded server". The control loop of a self-adaptive system is thus basically programmed as a set of such adaptation rules. Depending on the application requirements, the realization of these dynamic adaptations may be complex, and the number of adaptations may become unwieldy and therefore difficult to manage. On the

[∗]FNRS Research Fellow.

other hand, reuse of this system-specific adaptation knowledge across a family of related self-adaptive systems is possible and desirable [8].

While there is a well-known SPLE approach for static systems [19], there has been less focus on self-adaptive systems. Yet, it would be desirable to at least apply a domain engineering approach that focuses at the analysis and design of reusable adaptions. However, the engineering process to come to generic adaptations that are reusable across multiple applications is far from trivial. We believe the main reason is that there has been not much support for explicitly capturing context-dependent dynamism in variability models such as Feature Diagrams (FDs) [16]. Albeit more recent work on variability models acknowledges the concept of dynamic variability in SPLE, e.g. [18], key concepts in SPLE, such as *context* and *binding time*, are not well understood in the light of self-adaptive systems. This limits the applicability of current SPLE approaches in real adaptive scenarios.

For the purpose of this paper, we stick to the definition of *context* as *any piece of information which is computationally accessible* [11], and *binding time* as *the time at which the system is bound to a particular variant* [24]. With *static binding* we refer to the binding to a variant prior to the usage of the system, while *dynamic binding* is considered to be the binding that occurs during runtime [23].

The goal of the present paper is to take the first steps towards an extended research agenda including modelling dynamic variability as part of the domain engineering phase of SPLE: what are the most challenging problems concerning modelling dynamic variability and the fundamental concepts of context and binding time, and how these issues affect the research community. In this regard, we list six common research questions aimed at defining a common research agenda for tackling these problems with domain engineering (issues with application engineering are out of the scope of this paper). This set of questions originated from three distinct research cases that motivate the different issues related to modelling dynamic variability in self-adaptive SPLs. Most of this research is actually carried out in the context of the Belgian MoVES project,[1] fostering the collaboration among different Belgian universities.

The paper is structured as follows. Section 2 presents three different cases that each raise several issues w.r.t. modelling dynamic variability. From these separate accounts, we formulate common research questions in Section 3, which are used to bootstrap a first high-level research agenda in Section 4. Section 5 introduces related work and confronts it to our research questions. We conclude the paper in Section 6.

[1] More information at `http://prog.vub.ac.be/moves`

2 Motivating scenarios

We now elaborate on three different motivating scenarios that prove the lack of support for modelling dynamic variability in adaptive SPLs. In each motivating scenario, we discuss the issues that were experienced when performing a concrete case study as part of our research: context-aware cellphone systems, a web-based e-government application and runtime interactions in domotics systems respectively.

2.1 Context-aware cellphone systems

2.1.1 Overview

Currently, there is little explicit support for context awareness in traditional software engineering techniques and tools, which makes the development of these applications even more complex. We introduce an intelligent cell phone as an illustration of a context-aware system. Whereas traditionally a cell phone's action (e.g. to receive or to make a call) typically corresponds to a single behaviour, context-aware cell phones aim to have multiple behavioural variations associated to a single action. The choice of the appropriate variation is determined by the use context of the system. For instance, context-dependent variations can look as follows: if the battery level is low, ignore all phone calls except for contacts classified as VIP; if the user is in a meeting, redirect all calls and messages to the secretary; if there is a WiFi connection available, try to make phone calls or send messages via VoIP since this is cheaper for the user; if there is a GPRS connection available, try to send messages using TCP/IP also since this is cheaper.

2.1.2 Current achievements

In [5], we proposed the Context-Oriented Domain Analysis (CODA) model as a specialised approach for analysing, structuring, and formalising context-aware software requirements. In this work we identify a number of relationships that may exist among context-dependent adaptations. A context-dependent adaptation can include another adaptation which means that the applicability of the second adaptation is verified only if the first one is activated. Next, a context-dependent adaptation can conditionally depend on another adaptation. In this case, the applicability of the second adaptation depends on the result of the first adaptation, yielding a sequential execution. We finally introduce the notion of a choice point which is a variation point or context-dependent adaptation which has multiple adaptations associated to it. Optionally, one can associate a resolution strategy to deal with semantically interacting adaptations.

The CODA model proposes a solution that considerably differs from the existing design and programming tech-

niques to model runtime variations such as if-statements, polymorphism or design patterns. Using such techniques for context-awareness would lead to cluttered implementations (due to scattered context-dependent if-statements), combinatorial explosion of class definitions (when using polymorphism to model all the possible adaptations to the context), or to considerable infrastructural overhead (when using design patterns). By separating the system's default behaviour from the behavioural variations, the CODA model enables to make a clear distinction between the tasks of context reasoning, and dynamic binding of the variations.

2.1.3 Open research issues

Although the CODA model can help in tackling some of the challenges for modelling context-aware software, a number of challenging issues needs to be further explored.

In our approach, we assume that context-dependent variations primarily occur while the program is running. This approach gives the highest dynamicity but rises the issue on how requests that are currently being processed should be affected by the dynamic software update. A more exhaustive analysis should be done to determine what exactly should occur at runtime (context acquisition, context reasoning, variation binding, etc.) and what can be derived to earlier stages of the software development. The decision as to when the behaviour should vary has an impact on the design of the system (use of dedicated design patterns to achieve variability) and the choice of technology (implementation platform, programming language paradigm, execution environment).

Experience has pointed out that the evolution of relationships among context-dependent adaptations in the CODA model is error-prone since contradictions and cases of under-specification might seep into the specifications. This is mainly caused by the fact that a new adaptation can possibly interact with all existing adaptations in the system. Nevertheless, the solution for this issue is not to add extra information to the CODA diagram as this can dramatically diminish its understandability.

2.2 Web based eGovernment application

2.2.1 Overview

PloneGov is an open source project fostering the development of web based *eGovernment* applications and gathering around 55 international public organizations into 19 products [1]. All these products promote the cooperative development of applications and web sites targeted to public organizations and their citizens. The worldwide scope of PloneGov yields specific legal, social, political or linguistic aspects to deal with. All constrain the features required from a given product, hence the need for flexibility regarding product derivation. For now, we focus on one of PloneGov's products, namely PloneMeeting, a Belgian government-initiated project offering advanced meeting management functionalities to national authorities.

Central to PloneMeeting is the concept of *meeting* itself. The allowed states of a meeting are defined according to a workflow, which can be changed in the configuration. There is, however, no restriction as of when such changes may be made, e.g. *installation* time or *runtime*. Yet, changing the workflow at runtime might result in an inconsistent system since the states of already existing meetings might not be compatible with those of the newly selected workflow.

The Plone internationalisation initiative intends to provide a flexible mechanism to manage language selection and display. The so-called PlacelessTranslationService (PTS) is Plone's built-in translation management service. The PTS uses the language of the web browser to automatically determine the display language of the pages.

2.2.2 Current achievements

In previous work [4], we introduced the idea of using SPL principles to engineer the PloneGov project. Our conclusion showed a number of organisational and technical problems that had to be tackled. such as handling the distributed developers and managing the already existing variability.

In [14], we focused on the identification and modelling of the variability in PloneMeeting. Since no variability model formerly existed, the variation points had to be reverse engineered from stakeholders, developers and existing artefacts to enable the re-engineering of configurable artefacts. We therefore defined a reverse engineering process taking these various information sources as input and producing separate FDs for the different concerns we identified.

The most significant results we obtained so far are four modelling challenges identified during the variability reverse engineering of PloneMeeting [13]. The first one refers to the implicit modelling viewpoint underlying the variability modelling. The second one discusses the modelling of contextual elements whose availability is unpredictable. The third one focuses on the consistency between the FD and its constraints as they both evolve over time. The fourth one addresses the representation of large sets of features in a FD. The workarounds we proposed to tackle these issues still have to be systematically applied to concrete cases and properly assessed.

2.2.3 Open research issues

Apart from the workflow selection and browser-dependent translation aspect of Plone, the dynamic side of PloneMeeting has been disregarded is recent work. Although being

a rather static application, PloneMeeting still exhibits some dynamic, typically runtime, configuration alternatives.

As the language selection scenario shows, the changing environment requires some extra flexibility from the system to keep it displaying pages flawlessly. Since each web browser encodes the language differently, elicitating, scoping and modelling this contextual information is our first issue. Secondly, a recurrent issue was the classification of binding times and the identification of the proper time granularity. Although extensively covered in mainstream literature [12], solutions to these issues are still fragmentary. Thirdly, a more practical research issue was the selection of the most suitable means to identify and mark dynamically configurable variation points. Finally, we struggled to express runtime constraints conditioning the evolution of product configurations. Further investigations will tell whether existing solutions are comprehensive enough.

2.3 Runtime interactions in domotics systems

2.3.1 Overview

Domotics systems typically combine a wide range of features in the area of home control, home security, communications, personal information, health, etc. We will motivate the relevance of runtime context information when modelling interactions by exploring two scenario's for protecting the housing environment. The first scenario concerns a *fire control* feature that turns on some sprinklers during a fire and a *flood control* feature which shuts off the water main to the home during a flood. Turning the sprinklers on during a fire and flooding the basement before the fire is under control results in the house further burning down. The second scenario involves a *presence simulation* feature that turns lights on and off to simulate the presence of the house occupants and a *doorkeeper* feature which controls the access to the house and allows occupants to talk to the visitor. Obviously, we would like the doorkeeper not to give away the fact that nobody is home if there is an unidentified person in front of the door in order to prevent the house owners from a burglary. However, if the person can be identified and trusted, there aren't any problems. Both the basement being flooded and the fact if a person can be identified or not are conditions that are only available at runtime.

2.3.2 Current achievements

Domotics systems already have been introduced in a product line context elsewhere by Kang et al. [17]. What is missing in their FDs is that the dependencies and other relationships between features cannot be expressed in terms of runtime context information. As a result, runtime behavioural feature interactions (FIs) caused by runtime variability cannot be modelled. A *behavioural FI* is a situation where a feature that works correctly in isolation does not work correctly anymore when it is combined with other features. The fact that an interaction only might or might not occur depending on the runtime context at hand makes it a *runtime behavioural FI*.

In previous work [20], we proposed a conceptual model to enable the management of interactions so that knowledge about interactions can be shared and used in the course of system evolution. An important part of the conceptual model consisted of a concern hierarchy that resembles the feature hierarchy in FDs. In this model, we already introduced the notion of a *condition* to take into account certain runtime circumstances. A shortcoming of current FD approaches is the lack of support for modelling exactly this runtime context information. To the best of our knowledge, no appropriate formalism or methodology exists to reify information about runtime behavioural FIs, reason about them and enforce resolutions. A number of extensions to FODA have been proposed to express more complex feature relations, e.g. using propositional logic. However, we are not convinced that propositional logic can distinguish between all possible interpretations of runtime behavioural FIs. In [21], we argument for instance that traditional logic is not suited to represent the fire and flood control FI.

It is also important to realise that traditionally used mechanisms, such as e.g. prioritisation, are not always feasible for representing runtime behavioural FIs. Next to the fact that an interaction between two features with the same priority cannot be resolved, the relative priority of two features to one another can be different in varying circumstances. The latter is illustrated by the second domotics scenario from above where everything depends on the result of identifying the visitor.

2.3.3 Open research issues

Based on this need for modelling runtime context information relevant for FIs, we can identify the following open research issues. First of all, although it is easy to come up with the relevant context information for particular FIs, answering the question what context to model is a non-trivial problem. Secondly, we need to decide on how to model runtime context information. Coming up with a widely applicable methodology for modelling runtime context information poses an interesting challenge. One of the usual suspects here are propositional logic, but are these sufficient to express the possibly complex relationships including runtime context information? Moreover, it is not clear how we can express more complex interactions involving more than two features. Finally, we need to ask ourselves if this runtime context information should be part of the FD itself or should be specified in a separate, dedicated modelling lan-

guage, complementary to the FD. In the latter case, an obvious need arises for traceability links between the different models. For now, we want to leave this open for discussion (cfr. RQ2). Either way, we are convinced that this kind of knowledge is an important form of information that can be (re)used to manage runtime variability and therefore should be modelled.

3 Common research questions

After having presented three distinct cases, each of which identifies different modelling problems, we will derive from them a number of crosscutting research questions. The goal of the present section is thus to formulate a common research focus for the coauthors of this paper.

RQ1: How to determine what context drives dynamic change? To the best of our knowledge, there is currently no methodological support to determine which are the context variables that will have an influence on dynamic change and to determine the course of action to deal with a change of these context variables. For instance, in case of the mobile phone example (Section 2.1), the decision of whether the low battery level is a trigger for forwarding phone calls could reasonably be made by a project manager or developer. The decision that VIP phone calls should be able to circumvent this, however, needs to be taken on a higher level (it requires an infrastructure that lets users decide who VIPs are). Furthermore, the need for additional context information might emerge from the combination of several features, as shown by the presence simulation and doorkeeper features in the domotics example (Section 2.3).

Such a methodology could be inspired, for instance, by Kang et al. FODA's *context analysis* [16]. It could also be based on the Problem Frames approach [15], which aims at identifying physical context given a requirement, or the KAOS method [25] whose goal is to elicit requirements based on high-level goals. The output of this methodology would be (i) a set of relevant context info specifying what elements in the environment of the system are relevant, (ii) constraints specifying when adaptations must be performed (due to changes in the context), (iii) concrete actions specifying what adaptations must be taken (to deal with the contextual changes).

RQ2: How to explicitly model context-dependent adaptations and how to compose it with the FD? Part of this question is whether or not the context information and adaptation knowledge uncovered in the methodology needs to be incorporated into the FD. In the case of CODA, for instance, it appears that including all context information in the FD leads to a highly complex and unwieldy diagram. A more scalable approach might be to model this kind of information in separate diagrams and to trace them to the features that are concerned. At the same time, it might appear natural to consider environment variables such as *battery level low* in the FD.

RQ3: How do non-functional concerns constrain the execution of context-dependent adaptations? Performing dynamic adaptations should preserve the non-functional properties of the system. For instance, in the event of a dynamic change, the structural integrity and global state-consistency of the system have to be ensured. Other non-functional properties of interest are reliability, correctness or efficiency. In the scenario of PloneMeeting, for instance, an issue is how to *reliably* change a running workflow. In this case, we need to specify when it is safe to change a workflow so that it remains compatible with the workflows of already existing meetings. In the scenario of the context-aware cellphone, several variations may apply for the context conditions in which the phone calls are received or made, and therefore a *correct* integration between the variations and the cellphone's base behaviour should be ensured. Other type of constraints involve taking into account the dependencies and conflicts between different context-dependent adaptations

RQ4: How to specify constraints in order to avoid under-specification? Given constraints (e.g. binding time and runtime behavioural interaction constraints) have to be expressed and formalised in some way. Take, for instance, the interaction between the fire control and flood control features in the domotics system. With current constraint languages for FDs, we found it hard to express this kind of constraint in order to capture it and/or process it later. It is difficult to capture all possible context combinations in a generic way (instead of enumerating all context combinations) or referring to context information at all.

RQ5: How to map domain models to implementation-specific elements? Given a suitable notation for expressing constraints (see RQ2 & RQ3), these constraints are ideally expressed at a level that makes abstraction of specific context info related to a particular implementation technology. For example, in the case of PloneGov (Section 2.2), there are many different ways in which a browser communicates the user language to the web server. At the level of domain analysis and domain design, however, one would like to identify and reason about the desired user language as a context element that is independent from the particular implementation technology. Yet during domain implementation, a specification is needed that maps this abstract context element to the appropriate browser-specific information.

RQ6: How to classify context-dependent adaptations according to their context and binding-time? As our three cases from above already indicate, there is seemingly no consensus as to what different types of context-dependent adaptations can be supported by a self-adaptive SPL. One of the key issues here is that the relation between

the three concepts, that are at the heart of what we call self-adaptive SPLs, namely "dynamism", context and binding time is not clear. And so, it seems imperative at some point to classify the different types of adaptation, primarily, by their suitable binding times and contexts.

4 Towards a research agenda

Having stated the research questions, let us examine how they affect the classical SPLE process by Pohl et al. [19]. This leads us to a more concrete research agenda. The well-accepted SPLE process, consists of a domain engineering and an application engineering phase. In the domain engineering phase, the scope of the product line is decided, and a set of customisable components developed. The application engineering phase exists for each product that is to be delivered. Following a requirement analysis, it starts by configuring the product, i.e. deciding what goes into the product, and ends with integrating and testing it.

By mapping the research questions to the SPLE process, we are able to identify a set of concrete objectives that must be achieved in order to realise a suitable SPLE approach for self-adaptive systems. Before we proceed, note that an SPLE approach for self-adaptive systems is defined as an extension and not as a replacement of a classical SPLE approach. For example the design of reusable components that make up the technical infrastructure of self-adaptive system (sensors, effectors, monitors, planners,..) remains largely the same. The extension that a self-adaptive SPLE approach brings focuses mostly on the domain engineering of the control loop in a family of self-adaptive systems:

RQ1 To implement RQ1, one would need to extend the domain engineering phase by (i) a *context scoping activity*, that decides what part of the context must be monitored; and by (ii) a *context modelling activity* that explicitly captures the essence of the context-dependent adaptations in a model, so that it can be referred to.

RQ2 Complementary to RQ1, addressing RQ2 would lead to a *scalable structure* for relating the context-dependent adaptation knowledge to standard feature models.

RQ3 Addressing RQ3 would involve a *quality attribute analysis* [2] to determine the important non-functional requirements (performance, reliability, ...) and to identify *reconfiguration tactics* that aim at preserving these quality attributes in the presence of dynamic adaptations.

RQ4 We expect that the outcome of RQ4 will lead to the *selection or the improvement of existing modelling and constraint languages* supporting constraint specification at the different stages of the context modelling and integration process.

RQ5 In order to address RQ5, a *translation infrastructure* is necessary that bridges the gap between the concepts of the context models elicited during analysis, and the concrete artefacts of the implementation. Generally speaking, this translation infrastructure involves the mapping from the context models to system-specific sensors and actuators, but also involves connecting context elements to specific state properties of software components.

RQ6 This RQ is rather of conceptual nature, we hope that it will lead to a better understanding of key concepts and to a clearer terminology. It thus affects the whole SPLE process, albeit indirectly.

The application engineering process also needs to be extended with activities involving the analysis of required context-dependent adaptations and the configuration, integration and testing of these adaptations into a fully operational control loop. But as stated in the introduction, this paper has focused mostly on domain engineering, and leaves the study of issues with application engineering for future work.

5 Related work

Cheng *et al.* [3] propose a research roadmap focusing on the requirements, modelling, engineering and assurance activities of self-adaptive systems. For each of them, the inadequacy of existing techniques to support the development of reliable self-adaptive systems and the challenges ahead are systematically formulated. Out of their study, they notably conclude that the design time and runtime processes can no longer be dealt with separately and advocate SPLE as a possible opportunity to drive the development of self-adaptive systems. All the research questions we set forth go along the same line of research by further precising the issues self-adaptivity raises in SPLE.

Fernandes *et al.* [7] present UbiFEX, a modelling notation extending existing feature diagram languages with contextual information. The general feature model generated with UbiFEX is composed of a feature model, a context feature model with the associated activation expressions, and context rules binding the context and feature models together. UbiFEX also comes with a simulation tool checking the consistency of the produced models.

Desmet *et al.* [5], propose the Context-Oriented Domain Analysis (CODA) which is heavily inspired by the original Feature-Oriented Domain Analysis (FODA) [16] used in product-line development. It enforces software engineers to think of context-aware systems as pieces of basic context-unaware behaviour which can be further refined by means of context-dependent adaptations at certain variation points. A context-dependent adaptation is a unit of behaviour that

adapts a subpart of a software system only if a certain context condition is satisfied. Both this work and the one from Fernandes *et al.* should provide valuable solution elements to RQ2 and RQ3.

Hartmann *et al.* [10] introduce context variability models (CVM) to represent high-level context information of software supply chains. A CVM is a general classifier of the context of products that is expressed in a FODA like notation. The combination of the CVM and the SPL feature model results in a Multiple Product Line Feature Model where the dependencies between both models are expressed with `requires` and `excludes` links. They do not explicitly present their work as suited to self-adaptive or dynamic systems. Once adopted, the context configuration choices are immutable and do not lead to self-adaptive behaviours. Conversely, Desmet *et al.* and Hartmann *et al.* [7, 10] consider the dynamic evolution of the context and its impact on the model. The relevance of this work to our current research will therefore require further evaluations.

Lee *et al.* [18] propose an approach grouping features in binding units which are assigned binding times and binding states used to constrain the SPL configuration in a dynamic context. Their framework also provides a product reconfiguration process encompassing a context analysis, a reconfiguration strategy selection and a reconfiguration implementation phase. Their solution might notably offer means to clarify the issues outlined in RQ6.

Gomaa *et al.* [9] present a solution based on reconfiguration patterns for dynamic reconfiguration of software product families. Their reconfiguration patterns are based on UML collaboration and state diagrams. They focus on components and do not model contextual information nor provide explicit links with variability models. Nevertheless, their approach might be part of a solution to RQ5.

In contrast, Schmid *et al.* [22] propose a taxonomy of issues that can arise when migrating a system from development time (static) to runtime (dynamic) variability. They analyse the impact of dynamicity on the input of the processes, the processes themselves and the output of the processes, and delineate the required capabilities of the code base for each of them. Such a taxonomy might help us investigating solutions to RQ5.

Although promising, these solutions still call for systematic assessments and need to be augmented with thorough guidelines covering the steps going from the context scoping down to the implementation of self-adaptive SPLs. We hope the outcome of this analysis will generate meaningful results that will help us answering RQ1 and RQ4.

Finally, different research projects indicate the relevance of defining a roadmap when it comes to investigating dynamic variability. DiVA[2] will combine aspect-oriented and model-driven techniques in an innovative way to provide a new tool-supported methodology with an integrated framework for managing dynamic variability in adaptive systems. Their basic idea is to use models at both the design time and runtime level to manage dynamic variability in combination with aspect-oriented modeling techniques in order to tackle the issue of the combinatorial explosion of variants. Model-driven techniques are then used to automate and improve the creation of (re)configuration logic. The MUSIC project[3] is a European project intending to offer an open platform for the development of self-adaptive mobile applications. Among the expected results of this project are a methodology, tools and a middleware suited for software developers. MUSIC builds further on the results of the MADAM project[4] in which adaptation requirements of mobile applications were studied and a theory of adaptation was developed. A set of reusable adaptation strategies and adaptation mechanisms, based on dynamically reconfigurable component architecture was one of their main results. Compared to DiVA, the main variability mechanism in these two projects consists in loading different implementations for each component type of the architecture. The idea of the AMPLE project[5] is to holistically treat variability at each lifecycle stage by combining aspect-orientation and model-driven techniques (similar to DiVA). Obviously, different implementation possibilities exist for binding variation points in various development stages, e.g. at design, development, deployment or even at runtime. Implementation artefacts will not only include traditional program code but also runtime configuration and domain specific languages. Therefore, one of the most promising results will be AMPLE's variability framework with integrated tool support for each lifecycle stage. The careful study of the ongoing research in these projects appears to be imperative to evaluate the sustainability of the coming results of our research.

6 Conclusion

SPLE process support for dynamically adaptive systems is fragmented, although a well-known SPLE process for static systems already exists. Therefore, the main contribution of this paper is our list of six common research questions indicating limitations we observed related to modelling variability in self-adaptive SPLs and identifying the need for clarification of fundamental concepts such as context and binding time. Starting from this set of research questions, we defined a high-level research agenda in which we discuss the needed enhancements to the traditional SPLE process in order to achieve SPLE for dynamically adaptive systems.

[2] `www.ict-diva.eu`

[3] `www.ist-music.eu/MUSIC/about-music`

[4] `www.ist-music.eu/MUSIC/madam-project`

[5] `ample.holos.pt`

Acknowledgements

This work was partially funded by the Interuniversity Attraction Poles Programme, Belgian State, Belgian Science Policy, the FNRS, and the VariBru project of the ICT Impulse Programme of the Institute for the encouragement of Scientific Research and Innovation of Brussels (ISRIB).

References

[1] PloneGov. http://www.plonegov.org/.

[2] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley.

[3] B. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, editors. *Software Engineering for Self-Adaptive Systems, 13.1. - 18.1.2008*, volume 08031 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2008.

[4] G. Delannay, K. Mens, P. Heymans, P.-Y. Schobbens, and J.-M. Zeippen. PloneGov as an Open Source Product Line. In *Workshop on Open Source Software and Product Lines (OSSPL07), collocated with SPLC*, 2007.

[5] B. Desmet, J. Vallejos, P. Costanza, W. De Meuter, and T. D'Hondt. Context-Oriented Domain Analysis. In *6th International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT 2007)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, August 2007.

[6] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli. A survey of autonomic communications. *ACM Trans. Auton. Adapt. Syst.*, 1(2):223–259, 2006.

[7] P. Fernandes and C. Werner. Ubifex: Modeling context-aware software product lines. In *2nd International Workshop on Dynamic Software Product Line Conference*, Limerick, Ireland, 2008.

[8] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *IEEE Computer*, 37(10):46–54, 2004.

[9] H. Gomaa and M. Hussein. Dynamic software reconfiguration in software product families. In *Software Product-Family Engineering*, Lecture Notes in Computer Science, 2004.

[10] H. Hartmann and T. Trew. Using feature diagrams with context variability to model multiple product lines for software supply chains. In *12th International Software Product Line Conference*. IEEE Computer Society, 2008.

[11] R. Hirschfeld, P. Costanza, and O. Nierstrasz. Context-Oriented Programming. *Journal of Object Technology.* `http://www.jot.fm`, 7(3), March-April 2008.

[12] A. Hubaux and A. Classen. Taming time in software product lines. Technical Report Draft Version, University of Namur, June 2008.

[13] A. Hubaux, P. Heymans, and D. Benavides. Variability modelling challenges from the trenches of an open source product line re-engineering project. In *Software Product Line Conference (SPLC'08)*, 2008. To appear.

[14] A. Hubaux, P. Heymans, and H. Unphon. Separating Variability Concerns in a Product Line Re-Engineering Project. In *International workshop on Early Aspects at AOSD*, 2008.

[15] M. A. Jackson. *Problem frames: analyzing and structuring software development problems*. Addison-Wesley, Boston, MA, USA, 2001.

[16] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, CMU, November 1990.

[17] K. C. Kang, J. Lee, and P. Donohoe. Feature-oriented product line engineering. *IEEE Software*, 19(4):58–65, 2002.

[18] J. Lee and K. C. Kang. A feature-oriented approach to developing dynamically reconfigurable products in product line engineering. In *10th International Software Product Line Conference*, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[19] K. Pohl, G. Bockle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, July 2005.

[20] F. Sanen, E. Truyen, and W. Joosen. Managing concern interactions in middleware. In J. Indulska and K. Raymond, editors, *7th International Conference on Distributed Applications and Interoperable Systems*, volume 4531 of *Lecture Notes in Computer Science*, pages 267–283. Springer, 2007.

[21] F. Sanen, E. Truyen, and W. Joosen. Modeling context-dependent aspect interference using default logics. In *5th Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAMSE)*, 2008.

[22] K. Schmid and H. Eichelberger. From static to dynamic software product lines. In *2nd International Workshop on Dynamic Software Product Line Conference*, Limerick, Ireland, 2008.

[23] K. Schmid and H. Eichelberger. Model-Based Implementation of Meta-Variability Constructs: A Case Study using Aspects. In *Proceedings of VAMOS 2008*, pages 63–71, Essen, January 2008.

[24] M. Svahnberg, J. van Gurp, and J. Bosch. A taxonomy of variability realization techniques. *Software – Practice and Experience*, 35(8):705–754, 2005.

[25] A. van Lamsweerde. Goal-oriented requirements enginering: A roundtrip from research to practice. In *RE '04: Proceedings of the Requirements Engineering Conference, 12th IEEE International*, pages 4–7, Washington, DC, USA, 2004. IEEE Computer Society.