Comparitive Study of Variability Management in Software Product Lines and Runtime Adaptable Systems

Vander Alves, Daniel Schneider, Martin Becker
Fraunhofer IESE
Fraunhofer Platz 1, 67663 Kaiserslautern, Germany
<*first name*>.<*last name*>@iese.fraunhofer.de

Nelly Bencomo, Paul Grace
Computing department, InfoLab21, Lancaster University,
Lancaster, LA1 4WA, United Kingdom
{nelly, gracep}@comp.lancs.ac.uk

Abstract

Software Product Lines (SPL) and Runtime Adaptation (RTA) have traditionally been distinct research areas addressing different problems and with different communities. Despite the differences, there are also underlying commonalities with synergies that are worth investigating in both domains, potentially leading to more systematic variability support in both domains. Accordingly, this paper analyses commonality and differences of variability management between SPL and RTA and presents an initial discussion and our perspective on the feasibility of integrating variability management in both areas.

1. Introduction

Software Product Line (SPL) [15] and Runtime Adaptation (RTA) [35] have traditionally been distinct research areas addressing different problems and with different communities (e.g., SPLC and ICSR in the former area and Middleware in the latter). SPL deals with strategic reuse of software artifacts in a specific domain so that shorter time-to-market, lower costs, and higher quality are achieved. In contrast to that, RTA aims for optimized service provisioning, guaranteed properties, and failure compensation in dynamic environments. To this end, RTA deals mostly with dynamic flexibility so that structure and behaviour is changed in order to dynamically adapt to changing conditions at runtime.

Despite the differences, there are also underlying commonalities with synergies that are worth investigating across both domains. For instance, in terms of commonalities, both areas deal with adaptation of software artifacts: by employing some variability mechanism applied at a specific binding time, a given variant is instantiated for a particular context. Accordingly, the research community has recently begun to explore the synergies between these two research areas.

On the one hand, motivated by the need of producing software capable of adapting to fluctuations in user needs and evolving resource constraints [27], SPL researchers have started to investigate how to move the binding time of variability towards runtime [4, 11, 33], also noticeable in the research community with even specific venues, such as the Dynamic Software Product Line (DSPL) workshop at SPLC, currently in its second edition. On the other hand, motivated by the need of more consolidated methods to systematically address runtime variability, RTA researchers have started to investigate leveraging SPL techniques [24, 14, 10].

Nevertheless, in either case, a refined and systematic comparison between these two areas is still missing. Such comparison could help to explore their synergy with cross-fertilization that could lead to more systematic variability support in both domains.

In this context, this paper presents two key contributions:

- it analyses commonality and differences of variability management between SPL and RTA. We define variability management as the handling of variant and common artifacts during software lifecycle including development for and with reuse. We choose variability management because we see it as the common denominator for exploring synergies between SPL and RTA;
- it presents an initial discussion and our perspective on

the feasibility of integrating variability management SPL and RTA.

The remainder of this paper is structured as follows. Sections 2 and 3 briefly review conceptual models for variability management in SPL and RTA. Next, Section 4 presents comparison criteria and compares variability management between SPL and RTA approaches. Section 5 then discusses potential integration of SPL and RTA. Related work is considered in Section 6, and Section 7 offers concluding remarks.

2. Software Product Line Variability Management

There are different approaches for describing SPL variability, for instance Orthogonal Variability Model [38] and PuLSE's meta-model [7]. In this work, we comply with the latter, which has been applied in research and industrial projects for years. In particular, Figure 1 depicts the conceptual model for a SPL, with a focus on variability management. The figure is based on a simplified version of the meta-model proposed by Muthig [36], highlighting some parts of the instantiation process and the actors involved.

A SPL comprises a set of products and a SPL infrastrucutre developed in a specific domain. The first are developed by the application engineer, whereas the latter are developed by the domain engineer and are reused in more than one product. The SPL infrastructure consists of SPL assets, which in turn comprise a decision model and SPL artifacts. A special kind of PLAsset is the PLArchitecture, which represents the SPL reference architecture. SPL artifacts are generic SPL artifacts, i.e., they embed variation points which have an associated binding time and can be described according to a given mechanism. A decision model represents variability in a SPL in terms of open decisions and possible resolutions. In a decision model instance, known as Product Model, all decisions are resolved, which is used to instantiate a specific product from SPL artifacts [6].

A product consists of product artifacts in a given context. A product artifact is an instance of SPL artifacts and comprises Variants, which in turn are instances of variation points after these have been resolved by the decision model when the application engineer configures this model into the product model.

Binding time refers to the time at which the decisions for a variation point are bound [32]. Examples of binding time are pre-compilation, compilation, linking, load, or runtime. Traditionally, SPL has been used mostly without the latter binding time. Therefore, in those cases, with instantiation of the product, variability is bound and a specific running application is obtained. Variation points and decision models then do not persist in the generated product.

Another artifact that does not persist in the generated product is context. We adopt the general definition of context proposed by Dey et al. [18]:"*Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and the application themselves*". For example, context can be language or regulations. Although context is considered during domain analysis and is used by the decision model, it is not explicitly integrated into the instantiated product. Further, it is often not formally represented; rather, it is usually informally available to the domain and application engineers in non-computable form [17].

Since variation points, decision models, and context usually do not exist in the instantiated SPL products, switching from one product to another product is only possible at earlier binding times. The corresponding transition then is not from one variant to another, but from the design space to a variant and requires direct intervention of the application engineer.

3. Runtime Adaptation Variability Management

The domain model for runtime adaptation showing the concepts that are relevant to this paper is depicted in Figure 2. The domain model is based on our experience in developing Dynamically Adaptive Systems (DASs) in middleware research [10, 26]. We define a DAS as a software system with enabled runtime adaptation. Runtime adaptation takes place according to context changes during execution. Example of DASs we have developed at Lancaster University are the adaptive flood warning system deployed to monitor the River Ribble in Yorkshire, England [30, 29, 10]; and the service discovery application described in [16]. The figure serves as a conceptual model to help explain the description that follows.

A Reference Architecture addresses specific solution Domains, such as routing algorithms, networking technologies, and service discovery. The Reference Architecture is specified by the Domain Engineer. DAS will typically employ a number of System Variants, which are sets of Component Configurations. Different component configurations can result in different connection topologies (compositions) as well as in different "internal" behaviour (parameters). Dynamic adaptation is achieved via transitions between Component Configuration variations over time; for example, components being added, removed and replaced, as the DAS adapts based upon environmental context changes. In any case, every Component Configuration must conform to

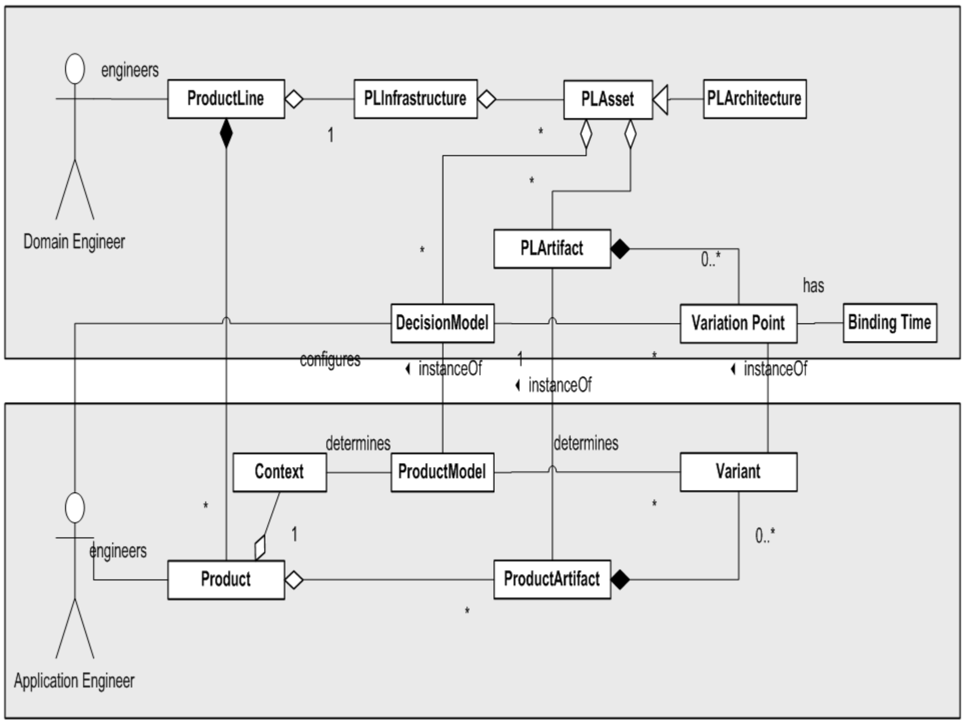

Figure 1. Variability Management in Software Product Lines.

the Reference Architecture, which describes the structural commonalities of a DAS that always hold.

In addition to models describing the configuration space at runtime, we require means to identify situations when to adapt and which configuration to choose. This is represented in Figure 2 by the transitions. A transition starts in a previous system variant and ends in a next variant. Transitions occur due to Triggers. Triggers are specified by the Application Engineer in terms of conditions of environment and context. We distinguish between two different types of approaches on how the best system variant can be determined [41]:

1. Rule-based approaches [10, 41, 43] usually have the "Event-Condition-Action" (ECA) form and hence distinctly specify when to adapt and which variant to choose. Such approaches are widely spread in the domains of sensor computing, embedded systems, and mobile computing. One reason is that such systems need to rely on light-weight approaches due to the inherent scarcity of resources (e.g., processing time, power supply) and must be deterministic. The rule sets are usually to be specified at design time. However, rules can also be added during execution [10].

2. Goal-based approaches [28, 34, 39] equip the system with goal evaluation functions in order to determine the best system variant under current circumstances. Neglecting available optimization strategies, the brute force approach would determine and evaluate all currently valid system variants and choose the variant that meets best the given goals. Thus, goal-based adaptation can be more flexible than rule-based adaptation and it is more likely that optimal configurations can be identified, albeit at a higher resource usage.

In our work, reconfiguration policies take the form of ECA rules. Actions are changes to component configurations while events in the environment are notified by a context manager. System Variants will change in response to changes of Environment Variants. Environment Variants represent properties of the environment that provide the context for the running system. Different values associated with these properties define the possible triggers of the transitions. System Variants represent different configurations permissible within the constraints of the DASs Reference Architecture. Environment Variants are specified by Domain Engineers. The variation points associated with the component configurations are specified using orthogonal variability models [38], which are not described in Figure 2 but addressed elsewhere [10]. For the DAS to operate in the context of any Environment Variant, it needs to be configured as the appropriate System Variant.

In our specific case, the dynamic reconfiguration is per-

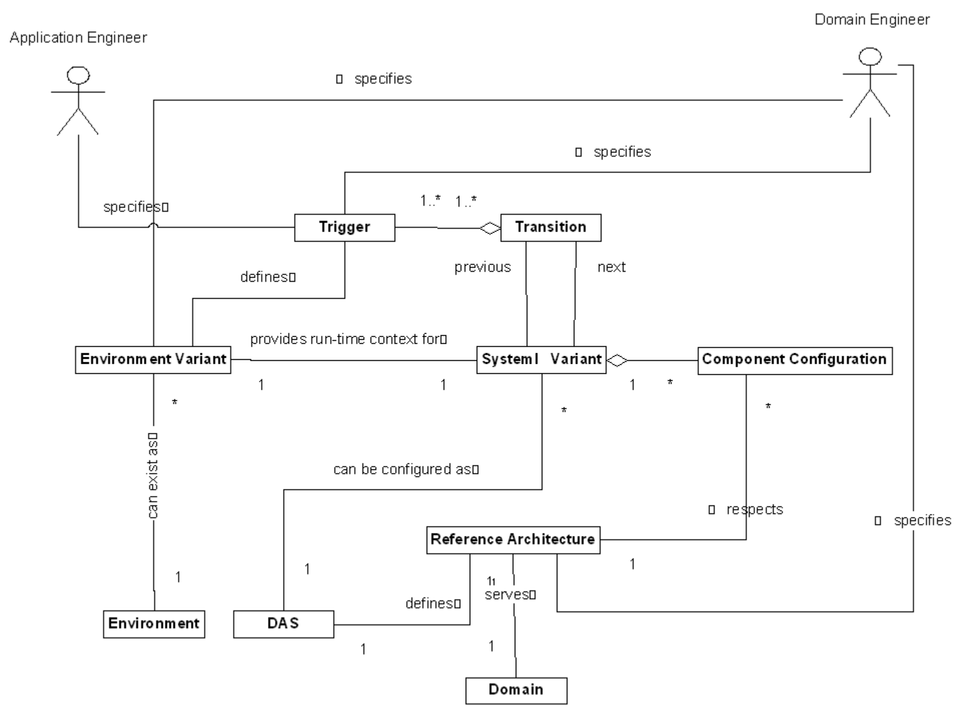

Figure 2. Variability Management in runtime adaptable system.

formed via the dynamic composition of components at runtime. However, the methodology is equally applicable to other runtime composition mechanisms, e.g., dynamic AOP.

In RTA, systems need to adapt to the prevailing situation on their own. Therefore there needs to be runtime models defining when and how to adapt. Further, there needs to be a notion of context, as acceptable configurations strongly depend upon the prevailing situation. We consider the following aspects as typical drivers for runtime adaptation: changes in required functionality and Quality of Service (QoS), including dependability, changes in the availability of services and resources, and occurrence of failures.

4. Comparison of Variability Management between SPL and RTA

After briefly presenting conceptual models of variability for SPL and RTA in Sections 2 and 3, respectively, we now compare both domains. First, we present comparison criteria (Section 4.1) and then perform the comparison (Section 4.2). The result of the comparison is later discussed in Section 5 for potential synergies.

4.1 Comparison Criteria

With the goal of identifying synergy points in variability management between SPL and RTA, the comparison criteria we use are based on previous work by Becker et al. [8] and by McKinley et al. [35], both in the context of RTA, and on the taxonomy proposed by Svahnberg et al. [40] in the context of SPL. The criteria are as follows:

- **Goal**: addresses the goal(s) of variability. If there is no goal or driver for variability, the rest of the criteria are irrelevant. Goals are generally expressed as statements over a variability driver. Some examples of variability drivers are the following: functionality, quality (dependability), resources (e.g., CPU, memory, communication, display, time, energy), context (e.g., regulation, culture, language). A goal, for instance, is to improve dependability or optimize resource usage;
- **Binding time**: time when the variability decisions are resolved, e.g., pre-compilation, compilation, linking, load, or runtime;
- **Mechanism**: a software development technique that is used to implement a variation point, e.g., parametric polymorphism, subtype polymorphism, design patterns, aspects, conditional compilation, reflection,

selection of one among alternative implementations, frames;

- **Who**: defines who resolves the variability decisions. It can be the domain engineer, the application engineer, the user, or the application itself;

- **Variability Models**: defines the models that have variation points or that control adaptation, e.g., decision model, reference architecture, PLArtifact, reconfiguration policies.

4.2 Comparison Results

According to the comparison criteria presented in the previous section, Table 1 compares variability management in RTA and SPL.

In terms of **goals**, although the fulfilment of functional and non-functional requirements is common in both SPL and RTA, SPL has primarily focused on providing fulfilment of functional requirements, whereas RTA has mostly focused on improving QoS while maintaining functional requirements. However, recently there has been a trend for SPL research to address QoS more closely [9, 22, 21, 24], which still remains an open issue and thus an explicit submission topic in key venues, e.g., SPLC09 [1].

Binding time in SPL has traditionally been at precompile, compile, and link time, whereas in RTA variability has been achieved at load time when the system (or component configuration) is first deployed and loaded into memory, and more commonly at runtime after the system has begun executing. The earlier binding in SPL usually allows for some static optimization and is thus usually more suitable for resource constrained systems, whereas the late binding time in RTA favours flexibility instead. Nevertheless, as mentioned previously, binding time in SPL has started to shift also to runtime in the context of DSPLs. Additionally, SPLs have also been built with flexible binding times, i.e., variation points that can have different binding times and binding times selected based on domain-specific context [12].

SPL **mechanisms** include diverse mechanisms such as conditional compilation, polymorphism, Aspect-Oriented Programming (AOP), Frames, parameterization, for example [3], and can be classified according to introduction times, open for adding variant, collection of variants, binding times, and functionality for binding [40]. On the other hand, at the core of all approaches to RTA adaptation is a level of indirection for intercepting and redirecting interactions among program entities [35]. Accordingly, key technologies are computational reflection, separation of concerns, and component-based design. Examples of corresponding techniques are Meta-Object Protocol (MOP) [31], dynamic aspect weaving, wrappers, proxies, and architectural patterns (such as the Decentralized Control reconfiguration pattern [25]). RTA mechanisms can be described according to the taxonomy by McKinley et al. [35], which highlights how, when, and where to compose adaptations. Key technologies and techniques for RTA variability can also be used for SPL variability, but in cases where runtime binding time is not required this leads to suboptimal resource usage, since variation points persist unnecessarily. Nevertheless, not all SPL variability mechanism can be used for addressing variability in RTA, e.g., conditional compilation. Additionally, SPL mechanisms allow transitions from PLArtifact to Product Artifact at early binding time, whereas in RTA transitions occur from component configuration to another component configuration at runtime.

In SPL it is the application engineer **who** is responsible for resolving and implementing variability decisions. This includes the instantiation of corresponding PLArtifacts (with aid of the decision model), the development of product-specific artifacts and their integration. The responsible entity in RTA depends on the actual binding time. It is an expert/user at load time and the system itself at runtime. Consequently, the system requires means to perform the role of the application engineer during runtime (and partially so at load time), when, due to context changes, reconfiguration is necessary so that a new variant is generated. As mentioned in Section 3, the application engineer in RTA only specifies the triggers, but does not actually perform the adaptation. Instead, triggers themselves play this role.

In terms of **variability models**, SPL involves using mechanisms to adapt PLArtifacts according to the decision model, whereas RTA mechanisms adapt System Variants according to the configuration models and the reconfiguration policies. These variants are then an instance of the Reference Architecture. RTA variability models are inherently available at runtime, thus requiring an explicit and computationally tangible representation of all such artifacts, whereas variability SPL artifacts in general do not have a runtime representation and are often expressed informally.

5. Analysis and Potential Synergy

Based on the comparison from the previous section, we can now highlight some commonalities between SPL and RTA variability management. This is essential to foster potential synergy and cross-fertilization of best practices in both research areas, which is feasible given the recent interest in DSPLs [27].

As mentioned in Section 4.2, the distinction between variability management **goals** of both areas has become blurred. Since SPL now addresses QoS more commonly, it could benefit from well-established techniques for guar-

Criteria	**SPL**	**RTA**
Goal	Focus on functional requirements	Focous on improving QoS while maintaining functional requirements
Binding time	Mostly Pre-process/Compile/Linking	Load time/Runtime
Mechanism	e.g., conditional compilation, polymorphism, AOP, Frames, parameterization	e.g., MOP, dynamic aspect weaving, wrappers, proxies
Who	Application Engineer	Expert/User, Application itself
Variability Models	Decision Model, PLArtifact	Reference archtecture, System Variant, Variability rules

Table 1. Comparison of Variability Management between SPL and RTA.

anteeing QoS at runtime that have been used in RTA. For example, Adapt [23] is an open reflective system that inspects the current QoS and then uses MOPs to alter the behaviour of the system through component reconfiguration if the required level of service is not maintained. Additionally, hybrid feature models, incorporating both functionality and QoS have also been proposed, e.g., by Benavides [9, 22, 21]. Conversely, RTA can use models for describing variability, such as enhanced versions of feature models [33] suitable for dynamic reconfiguration. Nevertheless, in this latter, there is still the challenge of addressing QoS issues [33].

Runtime binding time is on the focus of current research in SPL [33, 42, 4] and could leverage corresponding mechanisms in RTA. Wolfinger et al. [42] demonstrate the benefits of supporting runtime variability with a plug-in platform for enterprise software. Automatic runtime adaptation and reconfiguration are achieved by using the knowledge documented in variability models. Wolfinger et al. use the runtime reconfiguration and adaptation mechanism based on their own plug-in platform, which is implemented on the .NET platform.

In addition to the focus on runtime binding time in SPL, the *transition* itself towards runtime binding has also led to interest in **binding time flexibility**, whereby a variation point can be bound at different times [12, 19, 20]. The motivation is to maximize reuse of PLArtifacts across a larger variety of products. For instance, a middleware SPL could target both resource-constrained devices and high-end devices, and one variation point in this SPL could be the choice of a specific security protocol. For resource-constrained devices, small footprint size is more important than the flexibility of binding the variation point at runtime and thus the variation point is bound early with a specific security protocol. On the other hand, for high-end devices such flexibility is important and outweighs the incurred overhead (e.g., memory, performance loss due to indirection) of runtime binding of that variation point and thus the same variation point is bound at runtime depending on potential security threats or communication/throughput goals. Indeed, current research has proved the feasibility of implementing binding time flexibility, by using design patterns to make the variation points explicit and aspects to modularize binding time-specific code [12].

Although the relevance of binding time is well acknowledged [17], a concrete method for selection of the appropriate one and related to specific mechanisms is still missing. Such a method could leverage well-established practices in SPL and RTA, thus helping to explore their synergies.

Bindig time flexibility has increased the **importance of models** in RTA and SPL, e.g., DSPLs. For example, at earlier binding time, it is also important to model context in a more explicit and precise way, so that a *decison* about binding time can be made. Although acknowledged by traditional Domain Analysis, this has been represented informally and implicitly by the domain engineer. Conversely, in RTA, at later binding times, the decision model and context are still needed to decide on adaptation. Accordingly, for example, recent research also leverages the use of decision models at runtime [11]. Nevertheless, there remains the challenge of improving reasoning over this model at runtime. Conversely, the development of Reference Architecture in RTA could benefit from well established Domain Engineering approaches in SPL. This will help to discipline the process and leverage tested practices for building reusable artifacts. In particular, modern SPL component-based development processes such as Kobra [5] have features such as hierarchy model composition and refinement, and these could be enhanced with quality descriptions to be leveraged in RTA, thus helping to tame complexity.

The commonality among some models between SPL and RTA, the flexibility of binding time, and the blurredness of goals suggest that a holistic development process, exploring the synergies between SPL and RTA, would be beneficial to both domains. Particularly from the viewpoint of the RTA domain, there is still a general lack of appropriate engineering approaches. Accordingly, Adler et al. [2] introduced a classification with respect to the maturity of RTA approaches in three different evolution stages. In the state of the practice, adaptation is usually used implicitly, without dedicated models at development time or even at runtime (evolution stage 1). In the current state of the art some approaches emerged which use distinct models for variability and decision modelling (evolution stage 2). This naturally helps coping with the high complexity of adaptive systems by making them manageable, i.e., by supporting the modular and hierarchical definition of adaptation enabling the model-based analysis, validation, and verification of dynamic adaptation. The existence of a dedicated methodology enabling developers to systematically develop adaptive systems is considered as a further evolution step (evolution stage 3).

A holistic model-based engineering approach would naturally also benefit from the whole range of typical gains brought by model-driven engineering (MDE) approaches (i.e. validation, verification, reuse, automation). As for any other software engineering approach it is particularly possible to analyze and to predict the quality of the adaptation behaviour to enable systematic control of the development process. In our opinion, the combination of SPL and RTA approaches could bear a significant step in this direction. Further, the benefits of the combination would also include more consistent handling of variability across the binding timeline and leverage of modelling and analysis techniques across both domains.

6. Related work

Indeed, describing potential synergy between variability in SPL and RTA is not new. For instance, each system configuration can be considered as a product in a SPL in which the variability decisions necessary to instantiate the product are made at run-time [25]. Cheng et al. [13] present a roadmap for engineering Self-Adaptive Systems, where they suggest that technologies like: model driven development, AOP, and SPL might offer new opportunities in the development of self-adaptive systems, and change the processes by which these systems are developed. In contrast to these works, we explore this synergy in the context of our concrete experience in the SPL and DAS domains and highlight some points that lead to further research.

Gokhale et al. [24] propose an initial approach for integrating Middleware with SPL, focusing on the use of feature-oriented programming and model-driven development tools for uncovering and exploring the algebraic structure of middleware and handling runtime issues such as QoS and resource management. Classen et al. [14] identify limitations in domain engineering in current SPL research and propose a research roadmap for the integration of SPL and RTA, based on the key concepts of context, binding time, and dynamism. Similarly to these works, we highlight QoS challenges and the role of models, in particular context and decision model; in contrast, we additionally discuss challenges regarding binding time flexibility.

The availability of decision models at runtime is regarded as an essential property of the synergy between SPL and RTA. Anastasopoulos et al. [4] investigate the benefits of applying SPL in the context of the Ambient Assisted Living domain [37], in which systems have to be highly adaptive, proposing a roadmap for its use. As in our work, they identify the need to focus on runtime variability and to provide an execution environment that enables management and automatic resolution of decision models at runtime. Their work additionally proposes corresponding realisation techniques and a component model. Cetina et al. [11] propose a method for developing pervasive applications using SPL concepts and techniques. The decision model is represented at runtime and queried during system reconfiguration in order to address new user goals. Differently, we also identify challenges on achieving binding time flexibility.

7. Conclusion

We performed a comparative study of variability management between SPL and RTA with the aim of identifying synergy points and cross-fertilization opportunities that could lead to enhanced variability management in both domains. Based upon meta models for each of the two domains and a set of general classification criteria, we identified and discussed potential synergy points. From a SPL point of view, potential synergies comprise the specification and management of QoS and dependability properties, a more systematic approach towards variable binding time, and the formalization of context information and its relation to product variants and their properties. From the perspective of RTA, well-established variability modelling in the SPL domain promises to be a valuable basis for the definition of appropriate models at runtime as they are required in adaptive systems. We believe that addressing these synergy points would be best exploited by the definition of a holistic model-based engineering approach, which we plan to refine in future work.

Acknowledgements

The authors would like to thank Michalis Anastasopoulos for providing valuable feedback.

References

[1] *SPLC'09. Call for Participation.* `http://www.sei.cmu.edu/splc2009/files/SPLC_2009_Call.pdf`. Last access, Nov. 2008.

[2] R. Adler, D. Schneider, and M. Trapp. Development of safe and reliable embedded systems using dynamic adaptation. In *1st Workshop on Model-driven Software Adaptation M-ADAPT'07 at ECOOP 2007*, pages 9–14, Berlin, 2007.

[3] M. Anastasopoulos and C. Gacek. Implementing product line variabilities. In *SSR '01: Proceedings of the 2001 symposium on Software reusability*, pages 109–117, New York, NY, USA, 2001. ACM.

[4] M. Anastasopoulos, T. Patzke, and M. Becker. Software product line technology for ambient intelligence applications. In *Proc. Net.ObjectDays*, pages 179–195, 2005.

[5] C. Atkinson, J. Bayer, C. Bunse, E. Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Paech, J. Wüst, and J. Zettel. *Component-based product line engineering with UML*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[6] J. Bayer, O. Flege, and C. Gacek. Creating product line architectures. In *Third International Workshop on Software Architectures for Product Families - IWSAPF-3*, pages 197–203, 2000.

[7] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. DeBaud. Pulse: a methodology to develop software product lines. In *SSR '99: Proceedings of the 1999 symposium on Software reusability*, pages 122–131, New York, NY, USA, 1999. ACM.

[8] M. Becker, B. Decker, T. Patzke, and H. A. Syeda. *Runtime Adaptivity for AmI Systems - The Concept of Adaptivity*. IESE-Report; 091.05/E. 2005.

[9] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Automated reasoning on feature models. *LNCS, Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005*, 3520:491–503, 2005.

[10] N. Bencomo, P. Grace, C. Flores, D. Hughes, and G. Blair. Genie: Supporting the model driven development of reflective, component-based adaptive systems. In *ICSE 2008 - Formal Research Demonstrations Track*, 2008.

[11] C. Cetina, J. Fons, and V. Pelechano. Applying software product lines to build autonomic pervasive systems. In *SPLC '08: Proceedings of the 12th International on Software Product Line Conference*, pages 117–126. IEEE Computer Society, 2008.

[12] V. Chakravarthy, J. Regehr, and E. Eide. Edicts: implementing features with flexible binding times. In *AOSD '08: Proceedings of the 7th international conference on Aspect-oriented software development*, pages 108–119, New York, NY, USA, 2008. ACM.

[13] B. H. C. 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*. IBFI, Schloss Dagstuhl, Germany, 2008.

[14] A. Classen, A. Hubaux, F. Saneny, E. Truyeny, J. Vallejos, P. Costanza, W. D. Meuter, P. Heymans, and W. Joosen. Modelling variability in self-adaptive systems: Towards a research agenda. In *Proc. of the 1st Workshop on Modularization, Composition, and Generative Techniques for Product Line Engineering held as part of GPCE08*, October 2008.

[15] P. Clements and L. Northrop. *Software Product LinesPractices and Patterns*. Addison-Wesley, Reading, MA, 2002.

[16] C. F. Cortes, G. Blair, and P. Grace. An adaptive middleware to overcome service discovery heterogeneity in mobile ad hoc environments. *IEEE Distributed Systems Online*, 2007.

[17] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[18] A. Dey, D. Salber, and G. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction 16 (2)*, pages 97–166, 2001.

[19] E. Dolstra, G. Florijn, M. de Jonge, and E. Visser. Capturing timeline variability with transparent configuration environments. In *Proc. of International Workshop on Software Variability Management*, 2003.

[20] E. Dolstra, G. Florijn, and E. Visser. Timeline variability: The variability of binding time of variation points. In *Proc. of Workshop on Software Variability Management*, 2003.

[21] P. Fernandes and C. Werner. Ubifex: Modeling context-aware software product lines. In *Proc. of 2nd International Workshop on Dynamic Software Product Lines*, 2008.

[22] P. Fernandes, C. Werner, and L. G. P. Murta. Feature modeling for context-aware software product lines. In *SEKE*, pages 758–763, 2008.

[23] T. Fitzpatrick, G. Blair, G. Coulson, N. Davies, and P. Robin. Supporting adaptive multimedia applications through open bindings. In *CDS '98: Proceedings of the International Conference on Configurable Distributed Systems*, page 128, Washington, DC, USA, 1998. IEEE Computer Society.

[24] A. Gokhale, A. Dabholkar, and S. Tambe. Towards a holistic approach for integrating middleware with software product lines research. In *Proc. of the 1st Workshop on Modularization, Composition, and Generative Techniques for Product Line Engineering held as part of GPCE08*, October 2008.

[25] H. Gomaa and M. Hussein. Model-based software design and adaptation. In *SEAMS '07: Proceedings of the 2007 International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, page 7, Washington, DC, USA, 2007. IEEE Computer Society.

[26] P. Grace, G. Blair, C. Flores, and N. Bencomo. Engineering complex adaptations in highly heterogeneous distributed systems. In *Invited paper at the 2nd International Conference on Autonomic Computing and Communication Systems*, September 2008.

[27] S. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid. Dynamic software product lines. *Computer*, 41(4):93–95, 2008.

[28] S. Hallsteinsen, E. Stav, A. Solberg, and J. Floch. Using product line techniques to build adaptive systems. In *SPLC 2006: Proceedings of the 10th International Software Product Line Conference*, pages 141–150, Washington, DC, USA, 2006. IEEE Computer Society.

[29] D. Hughes, P. Greenwood, G. Coulson, G. Blair, F. Pappenberger, P. Smith, and K. Beven. An experiment with reflective middleware to support grid-based flood monitoring. *To appear in Wiley Inter-Science Journal on Concurrency and Computation: Practice and Experience.*

[30] D. Hughes, P. Greenwood, G. Coulson, G. Blair, F. Pappenberger, P. Smith, and K. Beven. An intelligent and adaptable flood monitoring and warning system. In *Proc. of the 5th UK E-Science All Hands Meeting (AHM06)*, 2006.

[31] G. Kiczales and J. D. Rivieres. *The Art of the Metaobject Protocol*. MIT Press, Cambridge, MA, USA, 1991.

[32] C. W. Krueger. Product line binding times: What you don't know can hurt you. In *SPLC*, pages 305–306, 2004.

[33] J. Lee and K. C. Kang. A feature-oriented approach to developing dynamically reconfigurable products in product line engineering. In *SPLC '06: Proceedings of the 10th International on Software Product Line Conference*, pages 131–140, Washington, DC, USA, 2006. IEEE Computer Society.

[34] G. Lenzini, A. Tokmakoff, and J. Muskens. Managing trustworthiness in component-based embedded systems. *Electron. Notes Theor. Comput. Sci.*, 179:143–155, 2007.

[35] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng. Composing adaptive software. *Computer*, 37(7):56–64, 2004.

[36] D. Muthig. *A Lightweight Approach Facilitating an Evolutionary Transition Towards Software Product Lines*. Fraunhofer IRB Verlag, Stuttgart, 2002.

[37] J. Nehmer, M. Becker, A. Karshmer, and R. Lamm. Living assistance systems: an ambient intelligence approach. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 43–50, New York, NY, USA, 2006. ACM.

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

[39] C. P. Shelton, P. Koopman, and W. Nace. A framework for scalable analysis and design of system-wide graceful degradation in distributed embedded systems. In *Words 2003: Proceedings of the Eighth International Workshop on Object-Oriented Real-Time Dependable Systems*, pages 156–163, Washington, DC, USA, 2003. IEEE Computer Society.

[40] M. Svahnberg, J. van Gurp, and J. Bosch. A taxonomy of variability realization techniques: Research articles. *Softw. Pract. Exper.*, 35(8):705–754, 2005.

[41] M. Trapp. *Modeling the Adaptation Behavior of Adaptive Embedded Systems*. Verlag Dr. Hut, Munich, 2005.

[42] R. Wolfinger, S. Reiter, D. Dhungana, P. Grunbacher, and H. Prahofer. Supporting runtime system adaptation through product line engineering and plug-in techniques. In *Seventh International Conference on Composition-Based Software Systems (ICCBSS 2008)*, pages 21 – 30, 2008.

[43] J. Zhang and B. H. C. Cheng. Model-based development of dynamically adaptive software. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 371–380, New York, NY, USA, 2006. ACM.