How dynamic is your Dynamic Software Product Line?

Nelly Bencomo and Jaejoon Lee
Computing Department, Lancaster University, United Kingdom
{nelly, j.lee}@comp.lancs.ac.uk

Svein Hallsteinsen
SINTEF ICT, Norway
Svein.Hallsteinsen@sintef.no

Abstract

Recently, there have been increasing demands for the postponement of decisions on software adaptations and product variations to provide the flexibility required by dynamic environments and users. The goal is that software adaptations and product variations can be chosen even at runtime. As such, a research theme that addresses development issues for reusable and dynamically reconfigurable core assets has emerged and it is called dynamic software product lines (DSPLs) with its consequential need to manage runtime variability. Research on the use of runtime variability, however, is still heavily based on the specification of decisions during design time. That is, a system simply postpones "when to adapt" to runtime but "how to adapt" is already decided at design time. In this paper, we present a brief assessment of the current research in the area and discuss some research issues related to the feasibility of DSPL oriented approaches to build self-adaptive systems.

Keywords: adaptation, runtime variability, self-adaptive systems.

1. Introduction

In [1], commonalities and differences of variability management between software product lines (SPLs) and runtime adaptation (RTA) are analyzed and initial discussions about the synergies and feasibility of integrating variability management in both areas are presented. One of the conclusions was that the SPL community can provide well established variability modeling techniques and software reuse, while profiting from the management of QoS and dependability properties, systematic approaches of variable binding time, and formalization of context information and relations with product variants. In this paper, we continue the research in those lines and present a brief report about different research works on runtime variability (also known as dynamic variability) [19]. Our main goal is to present a brief assessment of the current research to assess the feasibility of DSPL oriented approaches to build self-adaptive systems.

The remainder of this paper is organized as follows. Section 2 presents a range of approaches to tackle runtime variability. Section 3 provides a discussion about the comparison and contributions of the approaches and their abilities and shortcomings w.r.t. building self-adaptive systems. Some research issues and questions are also presented. Finally, Section 4 concludes the paper.

2. A range of approaches to tackle runtime variability

2.1 Criteria

Numerous researchers from different research areas and research projects have developed approaches that use the concept of runtime variability contributing to runtime adaptation of systems. Runtime adaptation deals with uncertainty about the environment and so changes (i.e. adaptations) cannot be fully predicted at design-time [34].

Our comparison criteria are related to two different dimensions of the uncertainty and are the following:

When to adapt: refers to the specific time where precise circumstances (context or environment) are given and provoke an adaptation or change. The specific moment of time is unknown during design time; however it could be characterized during design-time to eventually be solved at runtime.

How to adapt: refers to design decisions related to finding the appropriate sequence of actions to adapt and the software mechanisms that enable adaptation. Traditionally, in SPL and RTA techniques, the question *how to adapt* is decided by the designer, application engineer, user, i.e a human; however the final goal is having the system partially making (some) decisions.

It is important to highlight that in the context of this paper, the later these questions are answered the more dynamic the system is.

A further refinement of the criteria is provided by the MAPE-K model [21, 33] proposed by IBM as a reference model for autonomic computing systems. It models the autonomic element as consisting of an autonomic manager and a managed element. MAPE denotes the four main activities of the autonomic manager control loop (monitor, analyse, plan and execute), while the K denotes the knowledge about the managed element and its operating environment needed by these activities to perform successfully. Monitoring and analysis deal with "*when to adapt*" and planning and executing deal with "*how to adapt*".

We consider that the degree of dynamicity depends on to which extent these tasks are driven by models (the K component), and to which extent these models can be extended and refined at runtime.

When it comes to how to adapt, the kind of changes supported also plays an important role. At the basic level is setting configuration parameters, replacing individual parts and dynamic aspect weaving. At the next level is the ability to modify the architecture by replacing compositions. At the highest level is the ability to invent new compositions at runtime.

The surveyed approaches are presented and positioned in light of this model in the following sections.

2.2 Approaches

Geihs et al [14] from the team of the EU project MADAM promote the use of architectural models [13], use the adaptation capabilities offered by middleware platforms, and at some level treats dynamically adaptive systems as DSPLs with the corresponding support for variability management. In contrast to traditional event-condition-action (ECA) rules, they use goal policies provided during design time which are expressed as utility functions leaving to the running system the decisions on when to apply the actions required to implement the policies (tackling the question *when to adapt*). The reconfiguration steps to follow are determined by comparing the actual running system with new architectural variant models based on the utility function. The main variability mechanism provided by the MADAM approach consists of loading different implementations for each component type (primitive or composite) of the architecture. Using primitive component types, fine grain management of variability can be used to describe specific component replacements or specializations. They also take into account the benefits of coarse grained variability mechanisms using composite components. Both, fine and coarse-grained mechanisms are specified at design-time. The reasoning on when and how to adapt are based on models packaged with the component implementations (atomic or composite), and component implementations can be added or withdrawn at runtime. All variation points are open and the available component implementations and compositions are discovered dynamically at runtime. This enables to extend and refine the adaptation capabilities at runtime. The MADAM approach focuses on mobile computing applications.

The EU MUSIC project [32, 20, 23], strongly inspired in the previous results of MADAM, extends and generalises the experimental solutions developed in MADAM and take them to a next level of maturity [28]. It applies the same adaptation reasoning approach as MADAM, but generalises the approach to ubiquitous computing and Service Oriented Architectures (SOA). This includes extending the monitoring task with dynamic service discovery and the monitoring of service quality, and the executing task with the possibility to change service binding and service offers. With this the adaptation controller controls not only the internal configuration of a system, but also its collaboration with other systems, which in turn enables the creation of new structure at runtime. As in MADAM, in MUSIC all variation points are also open and the available component implementations and compositions are discovered dynamically at runtime. However, forming of new compositions at runtime is supported. The sequence of steps necessary to get from the current to the new configuration is derived at runtime based on the differences between the two.

Bencomo et al [3, 4] and the team of the UK project Divergent Grid [18] offer the Genie approach that also uses architecture models to support the generation and execution of dynamically adaptive systems based on component-based middleware technologies. A state machine is defined to specify the adaptive logic of the system. Each state embodies a system configuration (target system), and transitions describe when (needed conditions) and how the reconfigurations take place (reconfiguration script). The architectural models are complemented and documented using explicitly orthogonal variability models (OVM) [31]. These architectural and variability models are used to generate different software artifacts, e.g. configuration files and event-condition-actions (ECA) adaptation policies. The adaptation policies generated from the state machine solve the question *when to adapt* at runtime. The drawbacks of the approach are that (i) the possible system configurations need to be enumerated and fully specified, and (ii) the reconfiguration scripts should be written manually. However, using the API offered by the underlying execution platform some degree of support for unforeseen configurations and adaptation rules is managed as new behavior can be dynamically inserted [2] without restarting the system. New beha-

vior should also be validated in advance. Bencomo et al. work covers the domains grid, mobile computing, and embedded systems.

Wolfinger et al [35] prove the benefits of the integration of an existing product line engineering tool suite with a plug-in platform for enterprise software. In the same way as the Genie approach explained above, the knowledge documented in variability models support the decisions made to achieve automatic runtime reconfiguration and adaptation, i.e. design decisions contained in the variability models (decision models [10]) will be delayed till runtime (answering *when to adapt*). Different from the approaches Genie and MADAM/MUSIC described above, Wolfinger et al focus on enterprise software domains. While variability decisions in Wolfinger's approach are user-centered, the variability decisions in Genie and MADAM/MUSIC are mainly environment-centered.

Morin et al [27, 26] from the EU DiVA project [11] use model-driven engineering (MDE) and aspect-oriented modeling (AOM) techniques to support runtime variability. Tackling the explosion of the number of configurations potentially presented by all the approaches explained above, Morin et al offer an alternative solution that automatically builds architectures by composing modules (called aspects) associated to features, instead of fully specifying all the possible configurations. Depending on the current context, suitable aspect models are woven into a base model (that covers the communality), in order to build a complete target configuration, well fitted to the current context. After models validation, and as in MUSIC project, a comparison between the current configuration and the target configuration is performed. The comparison results allow the generation of the reconfiguration scripts needed to adapt the running system from one configuration to another more suitable to the current context. The reconfiguration scripts describe reconfiguration commands that adapt the system architecture enabling therefore the development of adaptive systems avoiding the enumeration of all the potential configurations (similar to MUSIC). Crucially, the adaptation model includes invariant properties and constrains that allow the validation of the adaptation rules before execution at design-time using model-checking techniques [12]. A set of adaptation rules are defined based on properties of the system-to-be optimized during runtime, i.e. adaptation rules express the adaptation policy in terms of the properties of the system to optimize and not directly in terms of the variants to use. Optimization decisions are performed at runtime (answering when to adapt). Variability decisions in Morin's et al approach are both environment-centered and user-centered contained in the specification of contexts.

The approach proposed by **Lee et** al is described in [25, 24]. The authors address issues in the area of adaptive service-oriented architectures (SOA) by adapting a feature-oriented product line engineering approach. The approach is based on the feature analysis technique to support the identification of services of service-oriented systems. The approach guides developers to identify services, to map users' context to relevant service configuration, and to maintain system integrity in terms of invariants and pre/post conditions of services. In their approach, the runtime system interacts with service providers' using an automated negotiation broker which uses QoS negotiation and service level agreement (SLA) evaluation and a provider rating to ensure service acceptability. A monitor is provided which actively observes the QoS requirements and triggers a new negotiation (adaptation) whenever SLA is violated at runtime (answering the question *when to adapt*).

Cetina et al [6, 7] from the SESAMO project outline reuse design variability models during runtime to tackle runtime variability. The benefits are immediate, as the design knowledge and existing model-based technologies can be reused at runtime. The runtime variability models support the self-reconfiguration of systems when triggered by changes in the environment. The question *when to adapt* is solved at runtime when checking a set of context conditions. The approach performs reconfiguration in terms of features and uses an engine called MoRE to translate contextual changes into changes in the activation/deactivation of features. The engine generates the reconfiguration plans (reconfiguration actions) that will modify the system architecture activating/deactivating features. Cetina et al have applied their approach to the smart-homes domain.

Perrouin et al [30] also tackle the problem of the exponential number of configurations and adaptations. However, different from works described above the authors acknowledges the complex relationships between the running system and its environment which may impact application functionalities and performance. Perrouin et al advocate that when capturing these relationships directly using a limited set of architectural models creates the risk of overlooking important environmental states and thus missing relevant architectural configurations. Therefore, the authors propose models for three different dimensions stressing separation of concerns: functional dimension (modelled using feature diagrams), topological dimension aiming at defining the possible bindings configurations for given

set of components (architecture), and platform dimension. The models of these dimensions are fully specified during design-time. Authors "*do not integrate timing issues*" [30], i.e. they do not tackle when to adapt, therefore all the design decisions are made before runtime and they are not re-evaluated at runtime.

Gomaa and Hussein [17] present their approach for the design of reconfiguration patterns for dynamic reconfiguration of software product families. Gomaa and Hussein see software reconfiguration pattern as a solution to the problem in a software product family where the configuration needs to be updated while the system is operational. Patterns define how a set of components participating cooperate to change the configuration of a system from one configuration of the product family to another. Their approaches do not explicitly say how to delay design decision till runtime.

There are other research projects and approaches; however, they tend to fail tackling the questions *when* and *how to adapt*, adding nothing new to our report.

3. Discussion

The overall results of the comparison are shown in **Table 1**. For each approach, the table shows if the questions are answered at runtime (using a check mark) or before i.e. at design-time.

Approach	When		How		
	M	A	P	E	K
MADAM	✓ at runtime and encoded in architecture/context model	✓ at runtime encoded in architecture/context model	✓ at runtime generation of reconfiguration scripts	✓ supports configuration parameters and replacement of components and compositions	context model+ architecture model annotated with property predictors and utility function. Models extendable at runtime
MUSIC	✓ at runtime and encoded in architecture/context model	✓ at runtime encoded in architecture/context model	✓ at runtime generation of reconfiguration scripts +dynamic service bindings (that were never designed explicitely)	✓ Same as above + service binding and service offers + aspects	Same as above + QoS aware model of available service providers and own service offers
Genie and Divergent Grid Bencomo et al	✓ at run-time and encoded in the state machine	X at design time and encoded in the state machine	X at design time and encoded in the state machine	X at design-time Component replacements Component compositions (with component frameworks)	context model+ architecture model Models extendable at runtime
Wolfinger et al	✓ at runtime Decision models	X at design time	X at design time	X at design-time Plug-in techniques	N/A
DiVA Morin et al	✓ at runtime adaptation rules	✓ at runtime Optimization of adaptation rules using fuzzy logic	✓ at runtime generation of reconfiguration scripts	✓ Component Compositions (architectural models) Aspect models/weaving	context model+ architecture model + aspects models Models extendable at runtime
Lee et al	✓ at runtime Monitoring SLA and QoS properties	✓ at runtime Optimization in terms of QoS	X at design-time Feature diagrams Service contracts	X at design-time Component replacements Component compositions (with C2 style architecture)	context model+ architecture model Models not extendable at runtime

SESAMO Cetina et al	✓ at run-time Monitoring context conditions	X at design-time	X at design-time Feature diagrams and Context properties	Feature tree (activation and deactivation of features)	Feature Trees Models not extendable at runtime
Perrouin et al	X at design-time	X at design-time	X at design-time Functional, topology and platform dimensions (UML models)	N/A	N/A
Gomaa and Hussein	X at design-time	X at design-time	X at design-time Design patterns (UML models)	N/A	N/A

✓ solved at runtime and driven by models/decisions specified at design time
(if extendable at runtime shown in K column)

X soved at designed-time, decisions are specified before runtime (design or deployment time)

Table 1. Results of the comparison.

The results show that there have been excellent contributions in the area. However, the survey also shows that our current DSPLs may not be as dynamic as we want to believe. The current research on runtime variability still focuses on the specification of decisions during design-time with few exceptions like generation of the reconfiguration scripts during runtime, (see discussions for MADAM, MUSIC, and DiVA). Decisions taken at runtime are heavily based on the uncertainty related to "*when to adapt*" and no definite answers are provided to tackle uncertainty related to "*how to adapt*". As an anecdote, it looks like we cannot get rid of the basic good old lessons from abstract interfaces and decomposition [29] and design by contract made during design. This may explain the bias we can observe in the research works towards the successful application of DSPLs and runtime variability on service-oriented domains (as in the case of MUSIC and Lee et al).

Among the different adaptation mechanisms that approaches use to support adaptation (i.e. *how to adapt)* are: Component replacements, Component compositions, Component frameworks, Plug-in techniques, Aspect Models, activation/deactivation of Features and Context properties, UML diagrams. In general, the mechanisms are specified during design-time and no explicit "ongoing design" support has been provided. Authors think that providing runtime support for *how to adapt* implies the use of runtime abstractions (i.e. models@run.time) that should be managed by the running system [5]. That way the running system would have access to consult and even change its own design. This is acknowledged by the K component of the MAPE-K model. The more knowledge the system is able to manage the more dynamic and adaptive the system can be.

Currently, replacing a composition with an alternative to some extent achieves changes of an ongoing design (see discussion of MUSIC). It is worth to say that the change of ongoing design (at runtime) should be controlled and bounded according to the domain and circumstances.

Open questions related to this are: *What kinds of runtime abstractions are meaningful to a DSPL? How suitable are the current SPL technologies to support this vision? How are these technologies suited to support the required new automation/adaptation process? Do we need new SPL technologies for this vision?*

An interesting result is the fact that software reuse, a basic term in the community of SPLs and Variability management, is not emphatically considered and explained in the approaches studied. The exceptions in the approaches studied are Cetina et al and Lee et al who discuss about the role of software reuse in their

approaches but not deeply. So far the SPL community is related to saving development efforts related to time and money (before runtime). Certainly, that meaning changes when decisions are delayed till runtime.

Open questions related to this are: *What does it mean software reuse at runtime? What kind of knowledge would we reuse? Who would get be the main beneficiary of such saving? developers? users? clients?* We think, the concept of software reuse needs to be reassessed under the new circumstances.

More research is needed to tackle the intrinsic uncertainty [34] involved in adaptive systems and delay design decisions related to *how to adapt* until runtime. The experience and results of the RTA community will prove again useful here [1, 9] as they have systematically developed mechanisms to support adaptations and late binding times [1, 8]. Work on artificial intelligence and bio-inspired mechanisms will also be relevant [22, 15]. However, this is not an easy task as we will need to deal with the consequent assurance that is required. Tackling this kind of uncertainty would mean dealing with higher risks and therefore the need to deal with assurance properties [36] that inevitably would need to be treated at runtime [16].

Open questions related to this are: *How well is the system meeting the requirements?, How to guarantee that the changes in the running system are correct? (e.g. with respect to the requirements), How to guarantee that the dynamic changes in the running system are performed correctly?* The usual collaboration that already exists between the SPL and requirements engineering (RE) communities will prove valuable here.

4. Conclusions

In this paper we have presented a brief assessment of the current research on DSPLs and runtime variability appraising their progress, achievements and downsides. Knowing our limitations will help us to improve our research. Other colleagues have done similar evaluation works [9].

The initial conclusions from our survey are that the current research on runtime variability is still heavily based on the specification of decisions during design-time. More research is needed to tackle the intrinsic uncertainty involved in adaptive systems and therefore the need to deal with assurance properties. Furthermore, leaving decisions about *how to adapt* to the running system would imply (i) the use of runtime abstractions (in such a way that the system is able to manage knowledge about itself and its environment) and (ii) the need to reassess the meaning of software and knowledge reuse of SPLs for adaptive systems. The authors on purpose avoid the use of DSPLs in the last sentence (to be provocative) as even though the community has achieved excellent results, it seems the survey implies that our DSPLs are not as dynamic as we want to believe.

For future research, we plan to extend the criteria of comparison and the research works evaluated. Software reuse is a candidate to be a criterion. Our original goal was to evaluate the research on DSPLs however, we are doing a similar evaluation for the case of the RTA community, i.e. we would like to know how dynamic the dynamic systems provided by the RTA community are. We are also interested in comparing the strengths and weaknesses of each community.

Acknowledgments: This work was partially funded by the DiVA project (EU FP7 STREP).

5. References

[1] Vander Alves, Daniel Schneider, Martin Becker, Nelly Bencomo, and Paul Grace. Comparitive study of variability management in software product lines and runtime adaptable systems. In *VaMoS*, pages 9–17, 2009.

[2] Nelly Bencomo and Gordon Blair. Using architecture models to support the generation and operation of component-based adaptive systems. In Betty H. C. Cheng, Rogerio de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee, editors, *Software Engineering for Self-Adaptive Systems*. LNCS, 2009.

[3] Nelly Bencomo, Gordon Blair, Carlos Flores, and Pete Sawyer. Reflective component-based technologies to support dynamic variability. In *2nd International Workshop on Variability Modelling on Software-intensive Systems (VaMoS'08)*, Essen, Germany, 2008.

[4] Nelly Bencomo, Paul Grace, Carlos Flores, Danny Hughes, and Gordon Blair. Genie: Supporting the model driven development of reflective, component-based adaptive systems. In *ICSE 2008 - Formal Research Demonstrations Track*, 2008.

[5] Gordon Blair, Nelly Bencomo, and Robert B. France. Models@ run.time. *Computer*, 42(10):22–27, 2009.

[6] C. Cetina, J. Fons, and V. Pelechano. Applying software product lines to build autonomic pervasive systems. pages 117–126, Sept. 2008.

[7] Carlos Cetina, Pau Giner, Joan Fons, and Vicente Pelechano. Autonomic computing through reuse of variability models at runtime: The case of smart homes. *Computer*, 42(10):37–43, 2009.

[8] Betty H.C. Cheng, Holger Giese, Paola Inverardi, Jeff Magee, Rogerio de Lemos, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Di Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. Software engineering for self-adaptive systems: A research road map. In Betty H. C. Cheng, Rogerio de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee, editors, *Software*

Engineering for Self-Adaptive Systems, volume 5525 of *Lecture Notes in Computer Science*, Dagstuhl, Germany, 2009. Springer.

[9] A. Classen, A. Hubaux, F. Saneny, E. Truyeny, J. Vallejos, P. Costanza, W. De 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 GPCE'08*, October 2008.

[10] Deepak Dhungana, Grünbacher Paul, and Rick Rabiser. Decisionking: A flexible and extensible tool for integrated variability modeling. In *VAMOS'07 First International Workshop on Variability Modelling of Software-intensive Systems*, 2007.

[11] DiVA. Diva-dynamic variability in complex, adaptive systems, http://www.ict-diva.eu/, 2008.

[12] Franck Fleurey, Vegard Dehlen, Nelly Bencomo, Brice Morin, and Jean-Marc Jezequel. Modeling and validating dynamic adaptation. In *Workshops and Symposia at MODELS 2008*, volume 5421M.R.V. Chaudron, 2008.

[13] Jacqueline Floch, Svein Hallsteinsen, Erlend Stav, Frank Eliassen, Ketil Lund, and Eli Gjorven. Using architecture models for runtime adaptability. *Software IEEE*, 23(2):62–70, 2006.

[14] K. Geihs, P. Barone, F. Eliassen, J. Floch, R. Fricke, E. Gjorven, S. Hallsteinsen, G. Horn, M. U. Khan, A. Mamelli, G. A. Papadopoulos, N. Paspallis, R. Reichle, and E. Stav. A comprehensive solution for application-level adaptation. *Softw. Pract. Exper.*, 39(4):385–422, 2009.

[15] Heather J. Goldsby, Betty H.C. Cheng, Philip K. McKinley, David B. Knoester, and Charles A. Ofria. Digital evolution of behavioral models for autonomic systems. *Autonomic Computing, International Conference on*, 0:87–96, 2008.

[16] Heather J. Goldsby, Betty H.C. Cheng, and Ji Zhang. Amoeba-rt: Run-time verification of adaptive software. In *Lecture Notes in Computer Science, Satellite Events at the Conference MODELS'2007 (Workshop Models@run.time)*. Springer-Verlag, 2008.

[17] Hassan Gomaa and Mohamed Hussein. Dynamic software reconfiguration in software product families. In *PFE*, pages 435–444, 2003.

[18] Paul Grace, Geoff Coulson, Gordon Blair, Laurent Mathy, David Duce, Chris Cooper, Wai Kit Yeung, and Wei Cai. Gridkit: Pluggable overlay networks for grid computing. In *Symposium on Distributed Objects and Applications (DOA)*, Cyprus, 2004.

[19] Svein Hallsteinsen, Mike Hichey, Sooyong Park, and Klaus Schmid. Dynamic software product lines. *IEEE Computer*, pages 93 – 95, 2008.

[20] Svein O. Hallsteinsen, S. Jiang, and R. Sanders. Dynamic software product lines in service oriented computing. In *3rd Int. Work. on Dynamic Software Product Lines (DSPL) (2009)*, 2009.

[21] Markus C. Huebscher and Julie A. McCann. A survey of autonomic computing—degrees, models, and applications. *ACM Comput. Surv.*, 40(3):1–28, 2008.

[22] Lee Jaejoon, Whittle Jon, and Storz Oliver. Bio-inspired mechanisms for coordinating multiple instances of a service feature in dynamic software product lines. In *Workshop Dynamic Software Product Lines (DSPLs)*, 2009.

[23] Shanshan Jiang, Svein O. Hallsteinsen, Paolo Barone, Alessandro Mamelli, Stephan Mehlhase, and Ulrich Scholz. Hosting and using services with qos guarantee in self-adaptive service systems. In *DAIS*, pages 15–28, 2010.

[24] Gerald Kotonya, Jaejoon Lee, and Daniel Robinson. A consumer-centred approach for service-oriented product line development. In *WICSA/ECSA*, pages 211–220, 2009.

[25] Jaejoon Lee, Dirk Muthig, and Matthias Naab. An approach for developing service oriented product lines. In *SPLC*, pages 275–284, 2008.

[26] Brice Morin, Olivier Barais, Gregory Nain, and Jean-Marc Jezequel. Taming dynamically adaptive systems using models and aspects. In *International COnference in Software Engineering (ICSE)*, 2009.

[27] Brice Morin, Franck Fleurey, Nelly Bencomo, Jean-Marc Jezequel, Arnor Solberg, Vegard Dehlen, and Gordon Blair. An aspect-oriented and model-driven approach for managing dynamic variability. In *MODELS'08 Conference*, France, 2008.

[28] MUSIC. http://www.ist-music.eu/music, 2008.

[29] D. Parnas. On the criteria for decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[30] Gilles Perrouin, Franck Chauvel, Julien DeAntoni, and Jean-Marc Jézéquel. Modeling the variability space of self-adaptive applications. In *2nd International Workshop on Dynamic Software Product Lines (DSPL 2008)*, pages 15–22, 2008.

[31] Klaus Pohl, GÃ¼nter Böckle, and Frank van der Linden. *Software Product Line Engineering- Foundations, Principles, and Techniques*. Springer, 2005.

[32] Romain Rouvoy, Paolo Barone, Yun Ding, Frank Eliassen, Svein O. Hallsteinsen, Jorge Lorenzo, Alessandro Mamelli, and Ulrich Scholz. Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments. In *Software Engineering for Self-Adaptive Systems*, pages 164–182, 2009.

[33] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4(2):1–42, 2009.

[34] Jon Whittle, Pete Sawyer, Nelly Bencomo, Betty H.C. Cheng, and Jean-Michel Bruel. Relax: Incorporating uncertainty into the specification of self-adaptive systems". In *17th IEEE International Requirements Engineering Conference RE 2009*, 2009.

[35] Reinhard Wolfinger, Stephan Reiter, Deepak Dhungana, Paul Grunbacher, and Herbert 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.

[36] Ji Zhang. *A formal approach to providing assurance to dynamically adaptive software*. PhD thesis, East Lansing, MI, USA, 2007. Adviser-Cheng, Betty H.C.