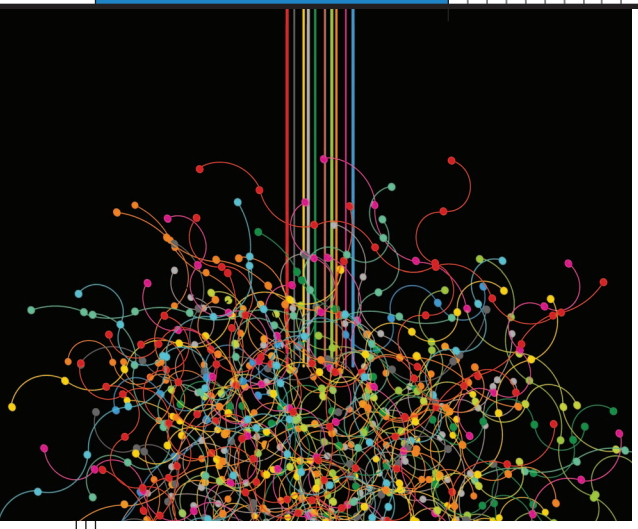

A View of the Dynamic Software Product Line Landscape

Nelly Bencomo, *Inria, France*
Svein Hallsteinsen, *SINTEF ICT, Norway*
Eduardo Santana de Almeida, *Federal University of Bahia, Brazil*

Dynamic software product lines extend the concept of conventional SPLs by enabling software-variant generation at runtime. Recent studies yield insights into the current state of the DSPL field, research trends, and major gaps to address.

01 Junier Amorim

As dynamic variations in user requirements and system environments become more frequent, there is a growing need for systems to be able to self-adapt to such conditions. To address this need, researchers have investigated dynamic software pr[illegible] lines (DSPLs).[1]

02 Junier Amorim

[illegible]s extend the concept of conventional SPLs by enabling software-variant generation at runtime. The reason for this extension is that typical SPL approaches do not fo[illegible]n dynamic aspects: the adapta[illegible]f their products is [illegible]rn and occurs during produc[illegible]e development. [illegible] the success of SPLs, DSPLs offer a promising strategy [illegible]al with the design and implementation of software changes that need to be handled at runtime.

3-6 4 notes:

Within this emerging field, however, practitioners and researchers still disagree about the terminolo[illegible] use and thus lack a common conceptual framework. [illegible]rmore, the study of DSPLs overlaps with that of other technologies, especially self-adaptive systems, leading to further confusion.

07 Junier Amorim

08 Junier Amorim

To better understand the DSPL landscape, we analyzed three complementary studies that used different methods to characterize the maturity of the field and identify major contributions and challenges. We synthesized the results of these studies to build a preliminary conceptual model. We also examined numerous other studies to identify state-of-the-art techniques and to highlight research trends and gaps requiring further exploration.

SURVEYING THE DSPL LANDSCAPE

The first study[2] analyzed commonalities and differences of variability management schemes between SPLs and self-adaptive (called *runtime adaptive*) systems with the aim of identifying synergy points and cross-fertilization opportunities that could lead to enhanced variability management in both domains. The comparison criteria included the

- goal of the variability,
- binding time of resolved variability decisions,
- mechanism that implements the variation points,
- stakeholder resolving the variability decisions, and
- models that define the variation points or control adaptations.

For SPLs, potential synergies include quality-of-service (QoS) specification and management, a systematic approach to managing binding time, and formalized context information and its relation to product variants. For self-adaptive systems, well-established variability modeling in the SPL domain shows promise as a way to define appropriate models at runtime.

The second study[3] examined the structure and maturity of the DSPL field. The authors explored several questions: What aspects of the DSPL have researchers already studied? Across what disciplines? What novel evaluation did the authors employ?

Published by the IEEE Computer Society

The third study[4] compared the feasibility of achieving runtime variability with current DSPL-oriented approaches. The researchers focused on two aspects of self-adaptive systems: *when* they adapt—the point at which specific contextual or environmental circumstances provoke a change; and *how* they adapt—the design decisions related to binding the appropriate adaptation action sequence and the software mechanisms that enable the adaptation.

DSPL CONCEPTUAL MODEL

09 Junier Amorim

To create a conceptual model for DSPLs, we combined an SPL model proposed in the first study[2] with the MAPE-K model for autonomic computing used in the third.[4]

10 Junier Amorim

The SPL model distinguishes two software engineering processes. *Domain engineering* produces the product line infrastructure, which consists of the common architecture and its variation points; a collection of reusable parts that fit into the architecture; and the decision model, which identifies variability and commonality in domain requirements and how requirements choices map to choices of parts.

11 Junier Amorim

Application engineering uses the product line's infrastructure to build products for particular contexts, represented by the context model.

The MAPE-K model derives its name from the main tasks in the feedback control loop typical of self-adaptive systems: *monitoring*, for detecting events that might require adaptation; *analysis*, for analyzing a change's impact on the product's requirements or constraints; *planning*, for deriving a suitable adaptation to cope with the new situation; and *executing*, for carrying out the adaptation. K denotes the *knowledge* necessary to perform these tasks in a useful way.

12-13 2 notes:

In a classic SPL, products are derived from the SPL infrastructure for a specific market or individual user and generally do not change after deployment. In a DSPL, as the lower part of Figure 1 shows, products can be reconfigured dynamically at runtime after their initial derivation, driven by the MAPE loop. Reconfiguration is based on models (the K element) maintained and represented according to classic SPL engineering practices, which means that parts of the SPL infrastructure are present at runtime. The use of SPL-inspired models and modeling practices is a characteristic DSPL feature.

14-15 2 notes:

16 Junier Amorim

The upper part of Figure 1 represents the DSPL equivalent of the domain engineering process—essentially, it is an additional instantiation of the MAPE-K loop that focuses on evolving the DSPL infrastructure, driven by feedback both from the deployed product and the application domain's general evolution.

17 4/12/19 Junier Amorim

In his keynote speech at the First International Workshop on Dynamic Software Product Lines in 2007, Klaus Schmid distinguished *bounded adaptivity*, which deals with context variation foreseen at design time and accommodated by DSPL models, from *open adaptivity*, which deals with context variation not foreseen at design time that requires model extension. The lower part of Figure 1 addresses bounded adaptivity, while the upper part addresses open adaptivity.

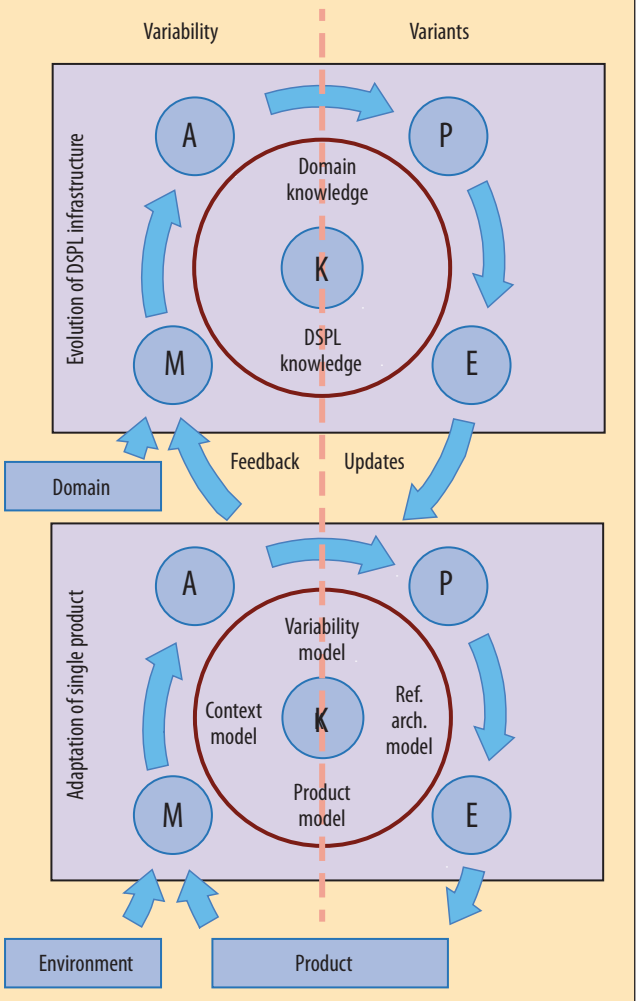

Figure 1. A dynamic software product line (DSPL) conceptual model. This model combines a classic SPL model with the MAPE-K model for autonomic computing.

MODELING RUNTIME VARIABILITY

At runtime, variations appear in the execution environment and in the product itself. Product variations depend on context variations, which can be identified by monitoring context properties. Here, we define context as information that is computationally accessible and describes both the execution environment and the product's current state. In self-adaptive systems, an adaptation to a given context corresponds to a product derivation at runtime.

Developers have used feature models and orthogonal variability models (OVMs) to specify product line variabilities, both at the requirements and design levels. Some

approaches use these models as a basis for modeling runtime variability in DSPLs.

Ildefonso Montero and colleagues[5] developed a feature model with execution time annotations to model business process evolution. Other researchers have focused on modeling runtime variability for context-aware systems.[6,7] The Genie tool[8] combines domain-specific languages with OVMs to model runtime variability supported by middleware platforms.

These efforts indicate that software reuse and variability management techniques in SPLs are useful for developing certain types of self-adaptive systems. However, more work remains to be done—for example, a DSPL's variability necessarily involves much more than the dynamic binding of variants identified in other variability models. The challenge for managing runtime variability is that it should support a wide range of product customizations under various circumstances that might not necessarily be known until runtime. Furthermore, as discussed later, recent studies have found that even new product variants can be identified only during execution.

DECISION MAKING

To automate adaptation decisions, it is necessary to model the relationships between the context's variation points and any product variants.

Rule-based approaches

Rule-based approaches perform the analysis and planning tasks at design time to produce a set of plans and drive adaptations in the form of event-condition-action (ECA) rules. An ECA rule includes triggering events, guarding conditions, current system configurations, and a set of reconfiguration steps. The idea is that when triggering events occur and a guarding condition holds, the system should perform the reconfiguration steps. The obvious benefits of this strategy include adaptation behavior analysis and validation at design time, and efficiency at runtime; on the downside, however, it requires the identification of all possible configurations and adaptations at design time.

Another problem with such approaches is that the number of rules can become unfeasibly large. The DiVA project[9] has addressed this issue by using a base model to describe configuration commonality and associate aspect models with variable features. After deciding which set of aspects to include, the developer weaves them into the base model. The system generates the necessary reconfiguration steps—that is, the ECA rules—at runtime by comparing the resulting product model with the model of the running product variant. Another promising rule-based approach combines a probabilistic machine-learning mechanism and offline training to establish the mappings; researchers have proven this approach's feasibility with small examples of DSPLs.[10]

Goal-based approaches

Goal-based approaches adopt more abstract representations of the decision model, allowing for analysis and planning at runtime.[11,12] They typically combine decision making based on the maximization of an objective function with explicit constraints to rule out invalid configurations.[13] Using goals avoids problems related to the enumeration of variants at design time, but at the cost of more runtime overhead.

One innovative goal-based approach uses the notion of claims to model assumptions that cannot be verified with confidence at design time.[12] The system monitors claims at runtime to test their veracity and, if a claim proves false, automatically selects an alternative goal realization, allowing the system to adapt to the prevailing environmental context. The system can also select a configuration not necessarily foreseen at design time. However, more work is needed to tackle this kind of adaptation using runtime variability management.

DYNAMIC CONFIGURATION

Klaus Schmid and Holger Eichelberger grouped runtime variability problems into six primary challenges:[14]

- deleting feature removal,
- reconfiguring components,
- transferring state,
- adapting interfaces,
- managing processing contexts, and
- dealing with references no longer available.

To tackle these challenges, researchers have combined product line ideas—parameterization, inheritance for specialization, and preprocessor directives—with runtime variability mechanisms such as component-based development (CBD)[15] and service-oriented architectures (SOAs). CBD uses component frameworks to design applications that can be adapted through reconfiguration. Developers achieve variability by plugging in different component implementations whose externally observable behaviors conform to a certain type. SOAs are effective for designing adaptable systems that can be reconfigured for contextual changes. Instead of conventional components, services do not have prescribed connectivity but are loosely coupled for dynamic use at runtime. SOAs allow for adding or removing features, fail-safe recovery, and re-adaptation based on resource changes.

Researchers have also investigated the feasibility of combining metamodels that incorporate runtime variability concepts with domain-specific languages.[9,16,17] ContextL,[18] for example, is a domain-specific language that uses the idea of layers to represent features whose activation state can be determined only at runtime; layers have IDs or names to which developers can add partial class and

method definitions. One study used domain-specific languages to model the solution architecture and the set of valid system configurations, and from these models automatically generated code and XML descriptions of component configurations.[16]

New techniques such as feature- and delta-oriented programming can provide more modularity for runtime adaptations. Instead of using assets with coarse granularity such as components, these techniques use features and deltas as first-class entities to improve reconfiguration of running products and check adaptation consistency.

OPEN ADAPTIVITY

Most works in the DSPL field focus on managing bounded adaptivity; not unexpectedly, product line engineering techniques are well suited to handle this. However, DSPL-based systems must also deal with the evolution of user needs and execution environments in ways not foreseen at the time of initial design. At a minimum, this requires changes and extensions to the design in terms of both functionality and adaptation capabilities.

To address this challenge, some researchers have proposed manual assistance and focused on mechanisms that support incremental artifact extension. For example, the authors of one recent study argued that DSPL products should support a management interface that enables extension propagation to the infrastructure at runtime.[19]

In general, goal-based approaches that perform analysis and planning at runtime based on explicitly represented models lend themselves more easily to incremental extensions. The MUSIC framework, for example, supports the dynamic addition and removal of component and service variants, as well as compositions with their own set of model fragments that describe internal variability.[20]

The self-adaptive systems community has extensively addressed evolving needs and environments—in other words, open adaptivity—but in a more automatic way at runtime.[21,22] The proposed approaches typically add an additional level to the adaptation control loop that monitors and adapts the implementation of the first level, often exploiting machine-learning mechanisms. In our conceptual model, this would correspond to automating part of the domain engineering process in Figure 1.

One example of open adaptivity from the DSPL community is an approach developed by Nadeem Abbas, Jesper Andersson, and Danny Weyns[10] that attaches to the MAPE loop the same learning mechanism used at design time to establish the decision model—the mechanism thus continues the decision model's evolution at runtime. However, seemingly only decision quality evolves, not the kind of adaptation the system is capable of or its functional capabilities.

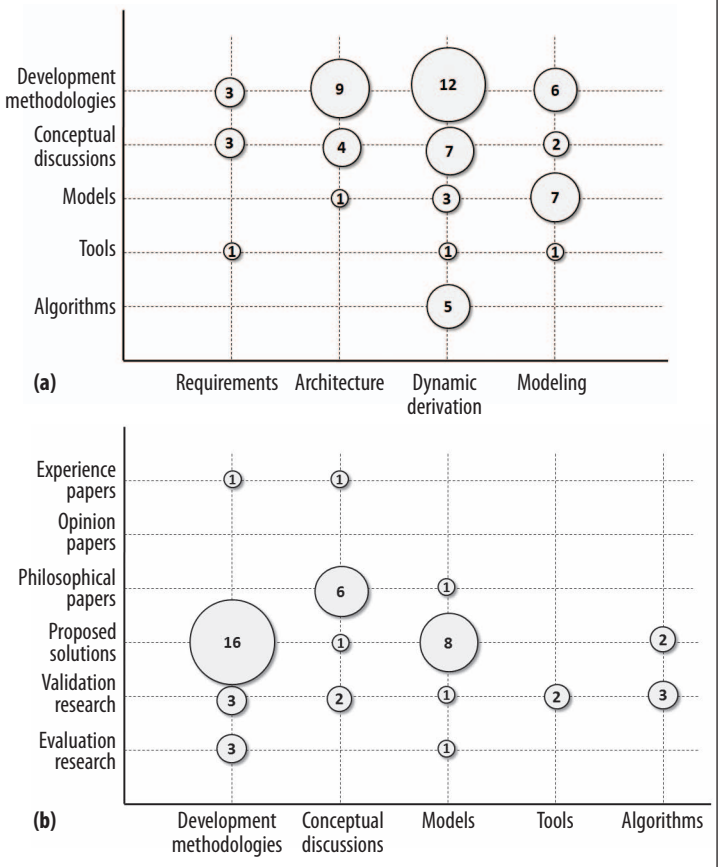

Figure 2. Research contributions to the DSPL field: (a) type and (b) maturity.

Also worth mentioning is Schmid and Eichelberger's notion of *metavariability*: modeling how the variability model may evolve.[14] Representing metavariability at runtime could be one way to better understand how to automate open adaptivity.

As with products derived from a traditional SPL, there can also be static differences among the contexts in which different instances of a configurable product are deployed. In a hybrid DSPL, the corresponding variation points would be bound permanently at initial derivation and not present at runtime. For example, EASy-Producer and Capucine both support a unified approach to modeling variability that decides at initial derivation which variation points are bound permanently and which are left open for rebinding at runtime.[14,23]

MATURITY OF THE DSPL FIELD

As Figure 2 shows, there are a significant number of contributions to DSPL research, primarily proposed development methodologies. To date, dynamic derivation is the most explored area, but there are also numerous studies on software architecture, which combines service orientation and adaptation, metamodels, and domain-specific

languages. Some areas still lack extensive work—for instance, algorithm development and tool support.

Figure 2 also indicates that the field is still in the formative stage—experience reports or evaluation research that demonstrate DSPL applicability in an industrial context are lacking. The same is true for tools and algorithms, which are largely in the validation stage and not in real industrial use yet. However, the prevalence of validation research across all types of contributions indicates the field's growing maturity. In addition, the presence of philosophical papers points to the software engineering community's interest in reaching common ground.

Researchers have proposed DSPLs for applications in a wide range of domains. For example, one study used dynamic reconfiguration to automate internal logistics in a transportation system's production and warehousing environments.[24] Another study described a DSPL approach to create a Web-based mobile tourist planner that lists a city's attractions, such as parks and museums, and recommends excursions based on weather and the user's current location.[6] Yet another study used DSPL techniques to develop a service robot for the elderly with user-controlled, map-based autonomous navigation.[25] Other application domains that could benefit from DSPLs include environmental monitoring,[13] automotive systems,[26] and smart homes.[27-29]

Research on SPLs' runtime and dynamic aspects first emerged at the Tenth International Software Product Line Conference in 2006, where several seminal papers[30-32] led to the creation the following year of the First International DSPL Workshop. In 2008, researchers began to explore architectural and dynamic configuration aspects of DSPLs, with some contributions on modeling, and by 2009, the interest shifted to dynamic derivation. Most recently, the field's focus has been on using novel programming techniques to boost modularity for runtime adaptation.

Researchers have addressed the need for runtime access to context, decision models, and product models in the form of runtime models.[33] The authors of one study[19] claimed that a DSPL needs runtime models to support the automated generation of software artifacts to promote dynamic evolution. They also argued for a runtime model of the product line architecture itself, as well as some form of state safety assurance.

There have also been advances in the area of runtime variability with respect to delaying the binding time to runtime. Researchers have been focusing on delaying not only decisions about variants but also the analysis involved. They propose to achieve this through goal models that avoid the early enumeration of all variants and allow the insertion of new variants at runtime. Goal models have also proved useful at tackling open adaptivity. For example, the system can use such models at runtime to select a new goal operationalization strategy not foreseen at design time—that is, to find new variants dynamically.[12] Researchers have also used goal models to model domain variability.[11]

Efforts to cope with dynamically varying contexts using SPL techniques have been fruitful as well. The many proposed solutions complement those in the area of autonomous and self-adaptive systems,[21,22] improving the software engineering community's overall research efforts. Many challenges remain, but the results of future efforts promise to take the next generation of DSPLs to new levels of autonomy, software reuse, and maintainability.

Acknowledgments

We express our gratitude to the many colleagues who have contributed to this research area. We also thank our coauthors in the three studies that motivated and formed the initial basis for this study. This work was supported by the EU Marie Curie Project Requirements@run.time, SINTEF and Telenor through their strategic research cooperation, and the National Institute of Science and Technology for Software Engineering (INES) in Brazil, funded by CNPq, FACEPE, and FAPESB.

References

1. S.O. Hallsteinsen et al., "Dynamic Software Product Lines," *Computer*, Apr. 2008, pp. 93-95.
2. V. Alves et al., "Comparative Study of Variability Management in Software Product Lines and Runtime Adaptable Systems," *Proc. 3rd Int'l Workshop Variability Modeling of Software-Intensive Systems* (VaMoS 09), ICB research report no. 29, Inst. for Computer Science and Business Information Systems, 2009, pp. 9-17.
3. V.A. Buregio, E.S. Almeida, and S.R.L. Meira, "Characterizing Dynamic Software Product Lines: A Preliminary Mapping Study," *Proc. 14th Int'l Software Product Line Conf.* (SPLC 10), Springer, 2010, pp. 53-60.
4. N. Bencomo, J. Lee, and S.O. Hallsteinsen, "How Dynamic Is Your Dynamic Software Product Line?," *Proc. 14th Int'l Software Product Line Conf.* (SPLC 10), Springer, 2010, pp. 61-68.
5. I. Montero, J. Pea, and A.R. Cortes, "Representing Runtime Variability in Business Driven Development Systems," *Proc. 7th Int'l Conf. Composition-Based Software Systems* (ICCBSS 08), IEEE CS, 2008, pp. 228-231.
6. G.H. Alférez and V. Pelechano, "Context-Aware Autonomous Web Services in Software Product Lines," *Proc. 15th Int'l Software Product Line Conf.* (SPLC 11), ACM, 2011, pp. 100-109.
7. P. Fernandes et al., "Feature Modeling for Context-Aware Software Product Lines," *Proc. 20th Int'l Conf. Software Eng. & Knowledge Eng.* (SEKE 08), World Scientific, 2008, pp. 758-763.
8. N. Bencomo et al., "Genie: Supporting the Model Driven Development of Reflective, Component-Based Adaptive Systems," *Proc. 30th Int'l Conf. Software Eng.* (ICSE 08), ACM, 2008, pp. 811-814.
9. B. Morin et al., "Models@Run.time to Support Dynamic Adaptation," *Computer*, Oct. 2009, pp. 44-51.

10. N. Abbas, J. Andersson, and D. Weyns, "Knowledge Evolution in Autonomic Software Product Lines," *Proc. 15th Int'l Software Product Line Conf.* (SPLC 11), ACM, 2011, article no. 36; doi:10.1145/2019136.2019177.
11. A. Lapouchnian and J. Mylopoulos, "Modeling Domain Variability in Requirements Engineering with Contexts," *Proc. 28th Int'l Conf. Conceptual Modeling* (ER 09), Springer, 2009, pp. 115-130.
12. K. Welsh, P. Sawyer, and N. Bencomo, "Towards Requirements Aware Systems: Run-Time Resolution of Design-Time Assumptions," *Proc. 26th Int'l Conf. Automated Software Eng.* (ASE 11), IEEE CS, 2011, pp. 560-563.
13. S. Hallsteinsen, S. Jiang, and R. Sanders, "Dynamic Software Product Lines in Service Oriented Computing," *Proc. 3rd Int'l Workshop Dynamic Software Product Lines* (DSPL 09), ACM, 2009, pp. 28-34.
14. K. Schmid and H. Eichelberger, "From Static to Dynamic Software Product Lines," *Proc. 2nd Int'l Workshop Dynamic Software Product Lines* (DSPL 08), Univ. of Limerick, 2008, pp. 33-38.
15. J. Floch et al., "Using Architecture Models for Runtime Adaptability," *IEEE Software*, vol. 23, no. 2, 2006, pp. 62-70.
16. N. Bencomo et al., "Dynamically Adaptive Systems Are Product Lines Too: Using Model-Driven Techniques to Capture Dynamic Variability of Adaptive Systems," *Proc. 2nd Int'l Workshop Dynamic Software Product Lines* (DSPL 08), Univ. of Limerick, 2008, pp. 23-32.
17. R. Froschauer, D. Dhungana, and P. Grünbacher, "Runtime Adaptation of IEC 61499 Applications Using Domain-Specific Variability Models," *Proc. 2nd Int'l Workshop Dynamic Software Product Lines* (DSPL 08), Univ. of Limerick, 2008, pp. 39-44.
18. P. Costanza, "Context-Oriented Programming in ContextL: State of the Art," *LISP50: Celebrating the 50th Anniversary of Lisp*, ACM, 2008, article no. 4; doi:10.1145/1529966.1529970.
19. M.A. Talib et al., "Requirements for Evolvable Dynamic Software Product Lines," *Proc. 14th Int'l Software Product Line Conf.* (SPLC 10), Springer, 2010, pp. 43-46.
20. S. Hallsteinsen et al., "A Development Framework and Methodology for Self-Adapting Applications in Ubiquitous Computing Environments," to appear in *J. Systems and Software*, 2012; http://dx.doi.org/10.1016/j.jss.2012.07.052.
21. B.H.C. Cheng et al., "Software Engineering for Self-Adaptive Systems: A Research Roadmap," *Software Engineering for Self-Adaptive Systems*, B.H.C. Cheng, ed., LNCS 5525, Springer, 2009, pp. 1-26.
22. R. de Lemos et al., eds., "Software Engineering for Self-Adaptive Systems: A Second Research Roadmap," to appear in *Software Engineering for Self-Adaptive Systems II*, R. de Lemos et al., eds., LNCS 7475, Springer, 2012; www.cs.washington.edu/homes/wuttke/resources/2012/lncs-roadmap-2.pdf.
23. C. Parra, "Towards Dynamic Software Product Lines: Unifying Design and Runtime Adaptations," PhD thesis, Dept. de Formation Doctorale en Informatique, Inria, 2011 (in French); http://tel.archives-ouvertes.fr/docs/00/58/34/44/PDF/thesis-parra.pdf.
24. A. Helleboogh et al., "Adding Variants on-the-Fly: Modeling Meta-Variability in Dynamic Software Product Lines," *Proc. 3rd Int'l Workshop on Dynamic Software Product Lines* (DSPL 09), ACM, 2009, pp. 18-27.
25. M. Kim, J.-H. Kim, and S. Park, "Tool Support for Quality Evaluation and Feature Selection to Achieve Dynamic Quality Requirements Change in Product Lines," *Proc. 2nd Int'l Workshop Dynamic Software Product Lines* (DSPL 08), Univ. of Limerick, 2008, pp. 69-78.
26. H. Shokry and M.A. Babar, "Dynamic Software Product Line Architectures Using Service-Based Computing for Automotive Systems," *Proc. 2nd Int'l Workshop Dynamic Software Product Lines* (DSPL 08), Univ. of Limerick, 2008, pp. 53-58.
27. R. Ali, R. Chitchyan, and P. Giorgini, "Context for Goal-Level Product Line Derivation," *Proc. 3rd Int'l Workshop Dynamic Software Product Lines* (DSPL 09), ACM, 2009, pp. 8-17.
28. J. Lee, J. Whittle, and O. Storz, "Bio-Inspired Mechanisms for Coordinating Multiple Instances of a Service Feature in Dynamic Software Product Lines," *Proc. 3rd Int'l Workshop Dynamic Software Product Lines* (DSPL 09), ACM, 2009, pp. 35-42.
29. C. Cetina et al., "Designing and Prototyping Dynamic Software Product Lines: Techniques and Guidelines," *Proc. 14th Int'l Software Product Line Conf.* (SPLC 10), Springer, 2010, pp. 331-345.
30. J. Lee and K.C. Kang, "A Feature-Oriented Approach to Developing Dynamically Reconfigurable Products in Product Line Engineering," *Proc. 10th Int'l Software Product Line Conf.* (SPLC 06), IEEE CS, 2006, pp. 131-140.
31. S. Hallsteinsen et al., "Using Product Line Techniques to Build Adaptive Systems," *Proc. 10th Int'l Software Product Line Conf.* (SPLC 06), IEEE CS, 2006, pp. 141-150.
32. Y. Wang et al., "PLA-based Runtime Dynamism in Support of Privacy-Enhanced Web Personalization," *Proc. 10th Int'l Software Product Line Conf.* (SPLC 06), IEEE CS, 2006, pp. 151-162.
33. G.S. Blair, N. Bencomo, and R.B. France, "Models@Run.time," *Computer*, Oct. 2009, pp. 22-27.

Nelly Bencomo *is a Marie Curie Fellow at Inria, Paris-Rocquencourt, France. She received a PhD in computing from Lancaster University, UK. Benecomo is a member of IEEE, ACM, and ACM SIGSOFT. Contact her at nelly@acm.org.*

Svein Hallsteinsen *is a senior research scientist at SINTEF ICT, Trondheim, Norway. He received an MS in applied physics and mathematics from the Norwegian University of Science and Technology. Contact him at svein.hallsteinsen@sintef.no.*

Eduardo Santana de Almeida *is an assistant professor of software engineering at Federal University of Bahia, Brazil, where he leads the Reuse in Software Engineering (RiSE) Labs. Almeida received a PhD in computer science from Federal University of Pernambuco, Brazil. He is a member of the IEEE Computer Society, ACM, and the Brazilian Computer Society. Contact him at esa@dcc.ufba.br.*

A View of the Dynamic Software Product Line Landscape

Bencomo, Nelly; Hallsteinsen, Svein O; de Almeida, Eduardo Santana

01 Junier Amorim Page 1
11/4/2019 16:05

02 Junier Amorim Page 1
11/4/2019 16:02

03 Junier Amorim Page 1
11/4/2019 16:02
the binding time does not occur during execution

04 Junier Amorim Page 1
11/4/2019 16:02

05 Junier Amorim Page 1
11/4/2019 16:04

06 Junier Amorim Page 1
11/4/2019 16:06

07 Junier Amorim Page 1
11/4/2019 16:04

08 Junier Amorim Page 1
11/4/2019 16:04

09 Junier Amorim Page 2
11/4/2019 16:07

10 Junier Amorim Page 2

11/4/2019 16:08

11 Junier Amorim Page 2

11/4/2019 16:08

12 Junier Amorim Page 2

11/4/2019 16:09

13 Junier Amorim Page 2

11/4/2019 16:13

14 Junier Amorim Page 2

11/4/2019 16:09

15 Junier Amorim Page 2

11/4/2019 16:12

16 Junier Amorim Page 2

11/4/2019 16:10

17 Junier Amorim Page 2

11/4/2019 16:16

The upper graph demonstrates a process of domain results receive

18 Junier Amorim Page 3

11/4/2019 17:59

19 Junier Amorim Page 3

11/4/2019 17:59

20 Junier Amorim Page 3

11/4/2019 18:00

21 Junier Amorim Page 3

11/4/2019 18:00

22 Junier Amorim Page 3

11/4/2019 18:01