Coping with Variability in Model-Based Systems Engineering: An Experience in Green Energy

Salvador Trujillo[1], Jose Miguel Garate[2], Roberto E. Lopez-Herrejon[3], Xabier Mendialdua[1], Albert Rosado[2], Alexander Egyed[3], Charles W. Krueger[4], Josune de Sosa[1]

[1] IKERLAN Research Centre. Mondragon, Spain
{strujillo, xmendialdua, jdesosa}@ikerlan.es
[2] Alstom Wind Power. Barcelona, Spain
{jose-miguel.garate, albert.rosado}@power.alstom.com
[3] Johannes Kepler University. Linz, Austria
{roberto.lopez, alexander.egyed}@jku.at
[4] BigLever Software. Austin, TX, USA
ckrueger@biglever.com

Abstract. Model-Based Systems Engineering (MBSE) is an emerging engineering discipline whose driving motivation is to provide support throughout the entire system life cycle. MBSE not only addresses the engineering of software systems but also their interplay with physical systems. Quite frequently, successful systems need to be customized to cater for the concrete and specific needs of customers, end-users, and other stakeholders. To effectively meet this demand, it is vital to have in place mechanisms to cope with the variability, the capacity to change, that such customization requires. In this paper we describe our experience in modeling variability using SysML, a leading MBSE language, for developing a product line of wind turbine systems used for the generation of electricity.

1 Introduction

In many domains, software engineering is but one discipline that contributes to the success of software systems. Indeed, software systems rarely stand alone but must be integrated into larger systems comprising specialized hardware and merging the expertise of a wide range of technologies –mechatronics, electrical engineering, aeronautics, etc. *Software-intensive systems* characterize such systems where software interacts with other software, systems, devices, sensors, and with people [1].

For software engineering, the development of software-intensive systems poses new challenges – from people who do not have the same technological background (which hampers their communication), to historical boundaries (development roles and responsibilities that have existed for a long time). In particular, in those domains where software has not been a vital part of a system but is increasingly becoming one, the introduction of software engineering has become

a source of confusion – as roles and responsibilities must shift but the various disciplines are not able to individually decide how to best do that. The role of software engineering is then often one of a "back-end" engineering discipline that is no longer able to make or affect important design decisions.

Model-Based Systems Engineering (MBSE) recognizes the new unifying role that modeling ought to take for the engineering of software-intensive systems, namely, providing a "common language" for communication among the multiple disciplines involved in the development of this type of systems. It should be noted though that this effort is by no means an attempt to replace the rich and expressive modeling concepts that exist and have proven invaluable in each of the distinct disciplines. On the contrary, the aim is to foster the understanding and exchange of information among all stakeholders. SysML is an example of a modeling language conceived to play that unifying role [2].

There is however one crucial need that has not been addressed in modeling languages such as SysML. Quite often, software-intensive systems need to be customized to fit the concrete and particular needs of different clients, users, developers, etc. The capacity of change that software artifacts must have to meet all the customization needs is collectively referred to as *variability* [3], and thus it must be readily expressed in modeling languages that aim to capture system customization demands.

In this paper, we present an approach to support variability in SysML and how it has been applied to an ongoing project on wind turbine systems for the generation of electricity. We summarize the lessons learned during the modeling and development of a family of wind turbine systems and what we believe are new venues of experimentation and research.

2 Systems Engineering

Systems Engineering has as its function to guide the engineering of complex systems [4]. It differs from traditional engineering in that it focuses on a system as a whole, that is, as a set of diverse and interrelated components that have complex relations amongst them. Next we provide the basic background on MBSE followed by a short description of the key characteristics of SysML used in our work.

2.1 Model-Based Systems Engineering

Model-Based Systems Engineering (MBSE) is defined by the International Council on Systems Engineering (INCOSE), as the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases [5]. This emerging discipline has evolved as a result of the increasing reliance on *Model-Driven Engineering (MDE)* technologies over the last decade [6]. Following the MDE philosophy, MBSE models

not only serve as documentation but can also be executed for simulations to further assist with the verification and validation of design decisions.

There has been an extensive research effort and applications of MBSE lead by INCOSE, OMG, and several other organizations. Many different methodologies have been proposed. A survey by Estefan describes the salient characteristics of some of them [7]. These surveys highlight the importance that using standards for the systems engineering modeling brings for tool vendors and users.

Similarly, there are several modeling languages and environments that have been developed and adopted for different types of application domains and user communities. An example of a proprietary environment for Model-Based design for dynamic and embedded systems is Simulink [8]. Another example is Dymola, used primarily in the automotive, aerospace, and robotics industries [9].

2.2 Systems Modeling Language Overview

The System Modeling Language (SysML) is a general-purpose modeling language for systems engineering applications [2]. SysML is an extension of a subset of the U*nified Modeling Language (UML)* [10]. SysML shares with standard UML the following behavior diagrams: sequence, state machines, and use case. In structure diagrams, SysML shares package diagrams and extends UML's activity, block definition, and internal block diagrams. SysML also adds two new types of diagrams: requirement, and parametric (an extension to internal blocks).

The basic structural units in SysML are *blocks*. They can represent hardware, software, mechanical parts, or other sorts of element that can constitute a system. There are two types of blocks: block definition diagrams describe the system hierarchy and system/component classifications, and internal block diagrams model the internal structure of a system in terms of its parts, ports, and connectors.

The requirements diagram describes the requirements hierarchies and their derivation. This diagram provides the means to relate a requirement to its corresponding model element(s). The parametric diagram capture constraints on system property values (e.g. physical constraints like weight) to help the integration of design models with engineering models.

Multi-View Modeling (MVM) is a common modeling practice that advocates that multiple, different and yet related models are required to represent the perspectives and information needs of diverse system stakeholders throughout the development process [11,12]. SysML is an example of MVM because of its distinct diagram types. A crucial issue in MVM is the expression and maintenance of the semantic relationships that exist amongst the elements in the distinct views.

3 Wind Turbine Systems and MBSE

The consumption of energy has dramatically increased in the last century, with the negative effect of incrementing pollution and CO2 emissions. Nowadays,

there is evidence that this prolonged consumption has caused significant impact on the environment. Most notably, the planet is facing a steadily complex problem derived from the global climate change known as Global Warning [13]. There are currently worldwide efforts to produce renewable energy with reduced impact on the environment. Wind energy is a major player in this market accounting for a significant rate of the renewable production. In some western European countries it has reached 20% market penetration with the aid of government subsidies [14].

The production of wind power is typically achieved by an array of wind turbines put together in a location called wind farm. The locations where wind farms are installed can vary significantly, for example in the sea, deserts, mountains, etc. Wind turbines are complex engineering systems that are composed of several mechanical elements and subsystems such as blades, pitch, rotor, generator, and current inverter. In charge of the turbines there is a control system that actually manages the entire power production process of a wind turbine system. The economical and ecological importance of wind power combined with the technological and engineering challenges of the wind turbine systems presents an ideal opportunity to exploit the substantial benefits of employing MBSE.

3.1 Why use SysML?

Wind turbine systems, as any complex system, have a large number of functional, performance, physical and interface requirements which have to be satisfied during the development process. This implies the need for a comprehensive requirements engineering and management during the project. The number of requirements in our systems are counted by the hundreds. The capacity of SysML to deal with large systems and the mature tool support available were deciding factors in our selection.

Structurally, wind turbine systems consists of various elements, like tower, nacelle, rotor, blades, blade pitches, sensors, actuators, generator, inverter, refrigeration subsystems, and so on. The control system alone consists of several main subsystems. These subsystems offer all kinds of information and interfaces. Their management alone is challenging because it typically involves teams from multidisciplinary backgrounds such as aeronautical (wind) engineers, electrical engineers, and software engineers. Additionally, our project involved geographically distributed project teams which demanded a common modeling representation and understanding. In this working environment, the ability to represent the system structurally, the modeling language expressiveness for all the participating engineering disciplines, availability of extensive documentation, and an increasing community of users were all crucial factors in favor of SysML.

Wind turbine systems, like many complex systems, must promptly adapt to changing environment conditions; in our case, conditions such as wind speed, direction, or temperature. This rapid adaption makes behavioral expressiveness a stringent requirement of a modeling language for this domain. The most remarkable challenge is the real-time demands imposed to effectively control the

conversion of wind energy into AC electricity and its transfer to the electric grid. SysML can meet these domain needs.

The ability to adapt the system design to respond to the needs of a customer with a reduced time-to-market is especially challenging in this sector due to the rapid pace of change in the wind power industry. Next we present the sources that originate the variability in our wind turbine systems to subsequently describe how they are addressed.

3.2 Sources of Variability

Wind turbine systems often need to be customized to meet the needs of different clients and locations worldwide. A one-size-fits-all system is inappropriate because of the large variability involved. In general, the sources of the variability in this domain are:

- Different power generation requirements. For instance, typical requirements range from 850 KW to 3 MW. As expected, to meet the demands on such a wide range, different topologies, technologies and consequently different system elements are needed.
- Different controlled elements. As an example, consider the alternatives for cooling parts of a system: refrigerated air, refrigerated water, or a combination of both. The existence of alternatives like these implies that the elements involved most likely differ.
- Different controlling strategies. For example, a system could be deployed in geographical locations with extreme weather conditions of cool, heat, humidity, etc. Consequently, the control behavior of the system elements may vary according to the deployment site.
- Different locations. In this case, countries or regions may have specific legal or environmental regulations that must be fulfilled.

All these sources of variabilities can describe the whole range of wind turbine systems available in the market. Nonetheless, like most product line systems, it is unfeasible to attempt at tackling all of them in a single product line. Thus a careful scoping (selection of which features to include in a product line) is required.

Figure 1 shows a feature model, rendered as a mind map for simplicity, of our scoped product line. The focus in this paper is on the generator subsystem of the wind turbine. For example, in this figure, `Cooling` denotes a mandatory feature as all generators do require a refrigeration system. The generators we considered can have three cooling alternatives denoted with the features: `Air_passive-Air_forced`, `Air_forced-Air_Forced`, and `Water-Air_forced`. A generator has only one of these alternatives installed, denoted with cardinality `1:1` in the figure. Next we present how these sources of variability were handled using variability management for our wind turbine systems.

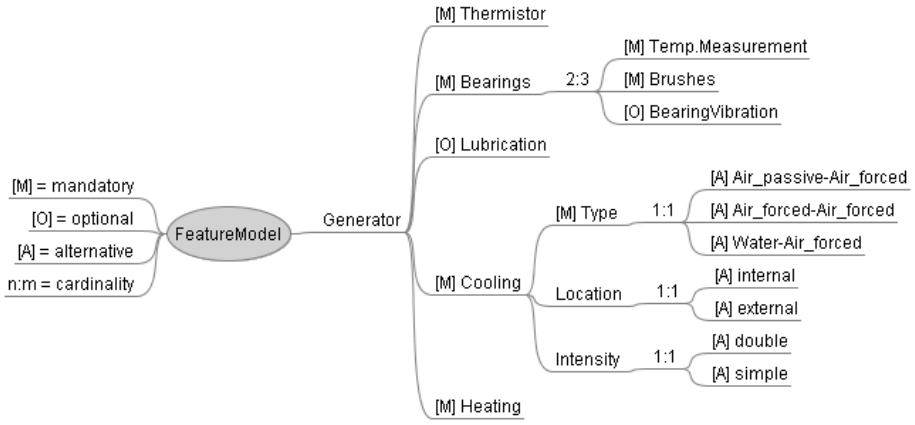

Fig. 1. Feature model - Partial subsystem

4 Variability Management with SysML

A *Software Product Line (SPL)* is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [15]. The significant benefits of applying SPL practices have been extensively documented and corroborated both in academia and industry [15,16,17]. Amongst them, the reduced time-to-market and the increased reuse of assets throughout the entire development cycle.

SPL approaches can be broadly categorized in two main groups depending on how they express variability in software artifacts. In *compositional* approaches, also known as with *positive variability* [18], the variable parts are encapsulated in modular units which are put together according to the features selected for building a system [19,20]. In *integrative* approaches, also known as with *negative variability*, the artifacts contain both the common and variable parts. Building a system means keeping the variable parts of the desired features in the artifacts while removing those parts belonging to unselected features [21,22].

Generally speaking, *variation points* are the places in the artifacts where variation can occur [23]. More concisely, a variation point is the representation of a variability subject within domain artifacts enriched by contextual information [17]. The context mentioned in this definition refers to the instantiation logic or mechanism to realize an actual artifact variant. In the case of integrative variability, the variation points and their instantiation logic are commonly denoted explicitly in the artifacts. An example of this are the *#ifdefs* macros of preprocessors.

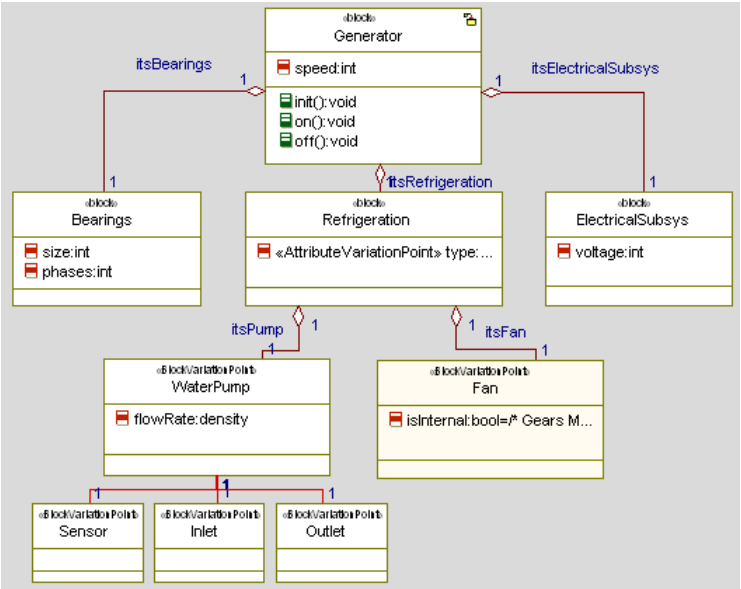

Fig. 2. Block Definition Diagram with variation points

4.1 Variation Points in SysML Diagrams

Variability is handled by using the notion of variation point of the BigLever Gears tool. System modeling is done by using IBM Rational Rhapsody. We handle variability in the diagrams by using the IBM Rational Rhapsody/Gears Bridge [24,25]. In short, these bridge tool provides support in three forms:

- Representation of variation points into model elements.
- Mappings among those variation-point-elements and features. These mappings are referred to as *feature logic*, and express the impact of features in model elements.
- Derivation of specific models according to feature selections, a process called *actuation*.

Engineering a system using SysML involves representing different diagrams, each providing a different perspective on the model such as structure or behavior. For example, Figure 2 shows a simplified block definition diagram for the generator subsystem of the wind turbine system. The upper side of this figure depicts the *blocks* `Generator`, `Bearings`, `Refrigeration`, and `Electrical subsystem`. These blocks corresponds to the major constituent elements of the generator subsystem. Some of these blocks are as well subdivided into further elements.

As highlighted before, our wind turbine systems form a family of products and as such they ought to be modeled. Consequently, variability must be reflected in the modeling diagrams. An example of this variability is the different refrigerating strategies; in our case, a system may be cooled using air or water.

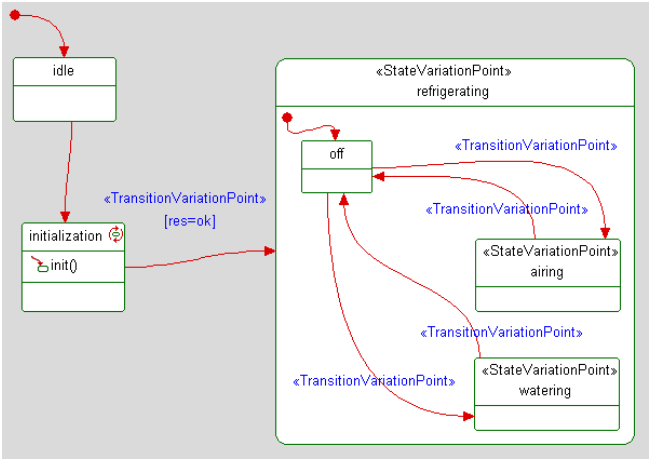

Fig. 3. Statechart diagram with variation points

Variability is expressed in SysML using stereotypes that represent the variation points of the different elements in the diagram. Figure 2 shows in block `Refrigeration` an `«AttributeVariationPoint»` to capture the variability of refrigeration strategies.

The `WaterPump` block models the water refrigeration strategy (in Figure 1 is feature `Water-Air_forced`), and the block `Fan` models the air strategy (feature `Air_forced-Air_forced` in Figure 1). In both cases, the entire block element is variable which is denoted with `«BlockVariationPoint»`.

Next we describe how actuation works. As a first step, the features desired for a particular member of the product line are selected. All common model elements, those without variation point stereotypes, will be part of the resulting diagram. The conditions associated to each element with a variation point (feature logic) are evaluated, if the conditions hold the respective element will be part of the resulting diagram. For instance, consider a wind turbine system that is refrigerated with water. For this option, feature `Water-Air_forced` is selected. The feature logic of the variation point of attribute `type` in `Refrigeration` block sets the initial value according to the selection. The feature logic of `WaterPump`, `Sensor`, `Inlet`, and `Outlet` causes these blocks to be included. In contrast, the logic of `Fan` will exclude this block when selecting `Water-Air_forced`.

Modeling variability at the family level, as opposed to each single system, fosters the reuse of system diagrams. But most importantly, it empowers the system designer with an enriched and new global perspective of all the design decisions and trade-offs involved. This perspective should not only address the needs of software engineers, but also those from the multiple disciplines usually involved in the conception and design of complex systems.

Diagram	Element	Stereotype
Block Definition *Internal Block*	Block	BlockVariationPoint
	Part	ObjectVariationPoint
	Attribute	AttributeVariationPoint
	Operation	PrimitiveOperationVariationPoint
	Port	PortVariationPoint
Statechart *Activity*	State	StateVariationPoint
	Transition	TransitionVariationPoint
	Action	ActionVariationPoint

Table 1. Elements supported in SysML

4.2 Variation Points in Multiple Views

Since SysML has different views, variation points are also available for their elements. For example, Figure 3 shows a simplified statechart for the refrigeration subsystem of the wind turbine system. The variation point mechanisms are the same as to those described for block diagrams. Besides statecharts and block definition diagrams, variability support is also available for internal block and activity diagrams. Table 1 summarizes the diagrams, their elements and the stereotypes to represent a variation point.

Note that Figure 3 specifies the behavior of the `Refrigeration` block in Figure 2. Notice here that the operations used within actions or transitions of a statechart should be defined in a block model. Such kind of dependencies among different views are common and make possible the communication among the distinct engineering disciplines. Hence the need of checking consistency of the different diagrams.

4.3 Perspective and Lessons Learned

The development of systems involve the combination of skills from different disciplines. Wind turbine systems are not an exception. Our team involves over two dozen engineers from software, embedded systems, mechanical and aeronautical backgrounds. This is an ongoing project that started two years ago. At present our focus is code generation from some models targeted to parts of the software architecture. Our preliminary results indicate an encouraging improvement in coping with the inherent system complexity. One of the reasons is that systems are now specified in terms closer to the problem domain, which gives engineers the ability to detect and resolve problems at that level while separating them from the software implementation details.

We summarize the lessons learned as follows:

– *Design as a continuum.* System design is a continuous process where each discipline focuses on certain design parts most frequently develop in parallel disciplinary teams. Introducing variability further accelerates this continuous process because changes may have a broader impact on the entire de-

velopment process. Having a common approach to handle variability across engineering disciplines plays thus a pivotal role.
- *Living with inconsistencies.* In this paper we assumed that the different system views should at anytime be consistent. However, there may be intermediate stages during the development in which certain inconsistencies may be allowed temporarily until some parts are completed by different teams.
- *Clearly defined system-software boundaries.* Recognizing the significance of separating systems from software engineering seems trivial. However, when considering the close relationships among different elements, it is far from obvious. This realization was a turning point in our project, and marked a first-step towards engineering our family of wind turbine systems. Our initial conception of variability shifted from a software-centric to a broader system perspective.
- *Software design driven by system design.* The ability to delay or lately modify design decisions is relevant in our case since the software have to fit the system elements it is controlling. Frequent situations of last-minute modifications in the system are now mitigated by the variability handling.

5 Related Work

There exist an increasing body of literature at the intersection of product lines and MBSE. In this section we present those pieces of work that most closely related to our approach and research experience.

Dauenhauer et al. describe an approach to model variability for testbed automation systems [26]. Contrary to our work, they use a positive variability approach whereby a models is constructed from a set of models through model transformations. Additionally, they define their own metamodel to represent and implement variability.

Beuche et al. present an approach for modeling binding-times using Simulink [27]. They define a metamodel for representing variation points in models of this language. In that sense, it follows a similar approach to ours because they also represent variability in terms of variation points that are made explicit in the models.

Favaro and Mazzini extend *FeatuRSEB*, a method for domain modeling, with SysML constructs [28]. This approach leverages SysML specific diagrams as follows: block diagrams to represent its so called context models and higher level architecture, requirements diagrams for its domain requirement models, and parametric diagrams to document its business decisions. One of the key differences with our work is that this extension to FeatureRSEB does make variation points explicit in SysML diagram elements.

6 Conclusions and Future Work

This paper described an approach to cope with variability in SysML modeling. We reported our experience in applying this approach as part of an ongoing

project on wind turbine systems for electricity generation. This application has proven successful because it treats system variability in an uniform way; in our case, this uniform treatment has improved of the overall development process.

Our work has also highlighted the need of mechanisms to ensure the consistency among the different SysML diagrams. We plan to leverage our experience and tool support in incremental consistency of UML models to address this pressing need [29,30].

Acknowledgments. This work was co-supported by the Spanish Ministry of Science & Innovation under contract TIN2008-06507-C02-02. This work is partially funded by the Austrian FWF under agreement P21321-N15. We would like to express our gratitude to Glenn Meter and Marty Bakal for their promptly support with the Gears/Rhapsody bridge. We would like to thank also to Oskar Berreteaga, David Gonzalez and Antonio Perez for their early advise.

References

1. Wirsing, M., Banâtre, J.P., Hölzl, M.M., Rauschmayer, A., eds.: Software-Intensive Systems and New Computing Paradigms - Challenges and Visions. Volume 5380 of Lecture Notes in Computer Science. Springer (2008)
2. Object Management Group (OMG): Sysml http://www.omgsysml.org.
3. Svahnberg, M., van Gurp, J., Bosch, J.: A taxonomy of variability realization techniques. Softw., Pract. Exper. **35**(8) (2005) 705–754
4. Kossiakoff, A., Sweet, W.: Systems Engineering. Principles and Practice. Addison-Wesley (2003)
5. International Council on Systems Engineering (INCOSE): Systems Engineering Vision 2020. Technical Report INCOSE-TP-2004-004-02 (September 2007)
6. Bézivin, J.: On the unification power of models. Software and System Modeling **4**(2) (2005) 171–188
7. Estefan, J.A.: Survey of Model-Based Systems Engineering (MBSE) Methodologies. Technical report (September 2007)
8. MathWorks: Simulink http://www.mathworks.com/.
9. Modelica: Dymola http://www.3ds.com/products/catia/portfolio/dymola.
10. Object Management Group (OMG): Unified Modeling Language (uml) http://www.uml.org.
11. Finkelstein, A., Gabbay, D.M., Hunter, A., Kramer, J., Nuseibeh, B.: Inconsistency handling in multperspective specifications. IEEE Trans. Software Eng. **20**(8) (1994) 569–578
12. Nuseibeh, B., Kramer, J., Finkelstein, A.: A framework for expressing the relationships between multiple views in requirements specification. IEEE Trans. Software Eng. **20**(10) (1994) 760–773
13. Intergovernmental Panel on Climate Change (IPCC): IPCC Assessment Reports http://www.ipcc.ch/.
14. World Watch Institute: Wind Power Increase in 2008 Exceeds 10-year Average Growth Rate (May 2009) http://www.worldwatch.org/.
15. Clements, P., Northrop, L.: Software Product Lines - Practices and Patterns. Addison-Wesley (2001)
16. van der Linden, F., Schmidt, K., Rommes, E., eds.: Software Product Lines in Action. Springer (2007)

17. Pohl, K., Bockle, G., van der Linden, F.: Software Product Line Engineering - Foundations, Principles and Techniques. Springer (2006)
18. Groher, I., Völter, M.: Using aspects to model product line variability. In: SPLC (2). (2008)
19. Mezini, M., Ostermann, K.: Variability management with feature-oriented programming and aspects. In Taylor, R.N., Dwyer, M.B., eds.: SIGSOFT FSE, ACM (2004) 127–136
20. Batory, D.S., Sarvela, J.N., Rauschmayer, A.: Scaling step-wise refinement. IEEE Trans. Software Eng. **30**(6) (2004) 355–371
21. Gomaa, H., Olimpiew, E.M.: Managing variability in reusable requirement models for software product lines. In Mei, H., ed.: ICSR. Volume 5030 of Lecture Notes in Computer Science., Springer (2008) 182–185
22. Zhang, H., Jarzabek, S.: XVCL: a mechanism for handling variants in software product lines. Sci. Comput. Program. **53**(3) (2004) 381–407
23. Jacobson, I., Griss, M., Jonsson, P.: Software Reuse: Architecture, Process and Organization for Business Success. Addison-Wesley Professional (1997)
24. Krueger, C.W.: Leveraging integrated model-driven development and software product line development technologies. In: OOPSLA '07: Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion, New York, NY, USA, ACM (2007) 836–837
25. Krueger, C., Bakal, M.: Systems and software product line engineering with sysml, uml and the ibm rational rhapsody biglever gears bridge. IBM White paper (2009)
26. Dauenhauer, G., Aschauer, T., Pree, W.: Variability in automation system models. [31] 116–125
27. Beuche, D., Weiland, J.: Managing flexibility: Modeling binding-times in simulink. In Paige, R.F., Hartman, A., Rensink, A., eds.: ECMDA-FA. Volume 5562 of Lecture Notes in Computer Science., Springer (2009) 289–300
28. Favaro, J.M., Mazzini, S.: Extending FeatuRSEB with Concepts from Systems Engineering. [31] 41–50
29. Egyed, A.: Instant consistency checking for the uml. In Osterweil, L.J., Rombach, H.D., Soffa, M.L., eds.: ICSE, ACM (2006) 381–390
30. Egyed, A.: Fixing inconsistencies in uml design models. In: ICSE '07: Proceedings of the 29th International Conference on Software Engineering, Washington, DC, USA, IEEE Computer Society (2007) 292–301
31. Edwards, S.H., Kulczycki, G., eds.: Formal Foundations of Reuse and Domain Engineering. ICSR 2009. In Edwards, S.H., Kulczycki, G., eds.: ICSR. Volume 5791 of Lecture Notes in Computer Science., Springer (2009)