Visual Language for Device Management in Telecommunication Product Line

Eugeny Semenov[1], Sheng Kai[1], Chen Gen[1], Dmitry Luciv[2,3], and Dmitry Koznov[2(✉)]

[1] Huawei Technologies Co., Ltd., Shenzhen, China
{semyonov.eugeny,kaisky.sheng,chengen}@huawei.com
[2] Saint-Petersburg State University, Saint Petersburg, Russia
{d.lutsiv,d.koznov}@spbu.ru
[3] Alferov University RAS, Saint Petersburg, Russia
dluciv@spbau.ru

Abstract. In the present paper, we consider the task of creating hardware specifications for telecommunication devices that close the communication gap between hardware engineers and software developers. This task has arisen during the development of a family of telecommunication systems at Huawei. The specifications need to describe hardware from the viewpoint of driver development (i.e., device management), omitting many hardware details and including information for automatic generation of driver data structures and signatures. Furthermore, they need to be comprehensive and illustrative for both engineering groups. To meet this challenge, we propose a visual language supporting five views (representations): Structured view (all structural elements of a device), Composite view (all connections of a device), Datapath view (device parts that process the data flow), Control view (device parts that control the data flow processing), and Service view (device parts that provide additional functionality). We present an implementation of the visual language built with the Eclipse Modeling Tools Xtext/EMF/Sirius and integrated into a development environment for device management. We have received positive feedback from the device management software engineers.

Keywords: Telecommunication systems · Control systems · Model-based approach · Product lines · Domain specific modeling · Domain specific programming

1 Introduction

Development of various telecommunication devices such as routers, firewalls, etc. is a very laborious task for a multitude of reasons. First, the final product includes very different, complex components, ranging from special hardware to control software. Second, companies that release such systems, usually, instead of a single product, create entire product lines [7] which encompass a variety of analogs, as

L. Bellatreche et al. (Eds.): MEDI 2021 Workshops, CCIS 1481, pp. 204–216, 2021.
https://doi.org/10.1007/978-3-030-87657-9_16

well as the new versions of already existing devices. Therefore, the development process includes a product line and concurrent work on several products, and can involve up to several hundred developers.

The complexity of the systems and their mass production produce large volumes of information. Therefore, establishing effective communication between teams is of utmost importance. A team should share information with other teams carefully: i.e., provide them with exactly what they require. At the same time, such "informational junctions" should support illustrative and convenient methods of specification representation to enable collaborative work between teams.

This problem has arisen in a project at Huawei that was developing network routers. During the interactions between hardware and software developers, a need was identified for hardware models of products that could be used in developing hardware drivers. This model needed to include high-level hardware descriptions, free from various hardware details. At the same time, it could contain additional information necessary for developing control software, which is a part of the final product. The model also needed to be complete, i.e., allowing automatic generation of essential parts of code for hardware drivers, as well as illustrative and easy to comprehend, discuss and develop.

We have decided to employ the domain-specific model driven approach [20, 32] and create a special visual language for this purpose. This language supports the multiple view point concept [25], allowing to model different aspects of the target products, and based on Domain Specific Modeling (DSM) approach [20]. We present an implementation of the language built with the Eclipse Modeling Tools Xtext/EMF/Sirius [13,31], and integrated into a device management development environment, specifically developed for this product line. We have received positive feedback from the device management software engineers.

2 Approach Context

In our work, we consider a particular product line of network routers. This product line consists of a set of systems (routers) that include both hardware and software components. In the current paper, we concentrate on a single task concerning the product line—device management. In short, device management refers to the development of drivers for the hardware included in the system.

The visual language that we present is a part of the device management development environment. The problem is that driver programmers need a large volume of information about hardware devices, but a still considerably smaller one than hardware engineers possess. Obtaining this information is a labor-intensive task, as hardware engineers are not able to easily extract information that is essential for software development from their body of knowledge. Moreover, it is important to formalize this extracted information in order to use it for automatic generation of software data structures, function signatures (parameters and their types), event sequence processing, etc. Generated code is an essential part of target code, although driver code can not be completely generated due to various peculiarities and specifics of particular systems.

To specify the hardware information used for driver development, we have developed a special textual device management domain specific language (DSL). The language allows to define not only the hardware structure of the system, but also a large number of its properties. Additionally, it provides facilities for specifying connections between hardware elements, which can be rather complicated due to a possibility of hardware structure changing dynamically: a system administrator can insert a special card into an existing router to increase the number of its network connections, and router drivers must obtain access to the new hardware elements on the card.

The visual language that we present is a part of the environment for driver development for the router product line. It is intended for graphical representation of the hardware structure and connections from the programming point of view, omitting secondary details. As it has turned out, the device management DSL is not enough for the product line needs. Our experiments have shown that quite often, software developers and hardware engineers need to discuss the structure of the system. Consequently, they need to use high level tools. Moreover, software developers are also needed to renew the information concerning the structure of a particular board or card due to the target system including decades of various. It should be noted that people operate with the visual presentation of the complex system structure much more efficiently than using other representations. However, it is more appropriate to perform everyday work using textual programming language [23].

3 Metamodel

To describe the proposed visual language, we use the metamodel approach [4,31]. This approach provides a formal method for language specification, and it can be used for graphical editor development. We used the Ecore notation [31]. Apart from some technical details, we present the simplified metamodel in the current paper (see Fig. 1).

Our visual language includes the main entities of the problem domain which are hardware elements **Module**, **Chip**, **Element**, **Port** and their connections. Hardware drivers can access these hardware entities and enable upper level software to influence their behavior (launch, re-launch, re-configure, etc.).

Let us describe in more detail the constructions of the language following Fig. 1.

HardwareElement is the root entity, denoting a hardware element that could contain ports. Its properties are *name*, *type*, and *multiplicity*, where the latter denotes the number of instances of this hardware element in a system. Instances can not be created dynamically and thus are statically presented in the system model. **HardwareElement** is virtual entity, and is introduced only for gathering properties of real life hardware elements.

Module represents a hardware part of the target router. There are two kinds of modules: *board* and *card*. Board is the main part of the router holding hardware elements which support the main functions of a router. Furthermore, a

board enables communication between all of the hardware elements placed on it. Card is used to extend the capabilities of a board or another card to be inserted into a target module. In our visual language, we provide the module types, but not the target configuration of the system, which can be modified dynamically.

Chip addresses the next layer of hardware decomposition: a module consists of a number of chips. Chips perform various module functions, e.g. coding/decoding an optical signal, board temperature control, etc. There are four kinds of chips: *core*, *slot*, *to_slot*, *to_bus*. Core chips are used to express main functionality of the module they belong to. Other chip kinds are used to express the mechanism of card insertion. Any module (*board* or *card*) could have a number of chips marked as a slot for various cards to be inserted in. If a module is a card, it must have a special chip marked as *to_slot* for insertion of the parent card into any suitable slot of another module (see chip `PicSlot` in Fig. 2). At last, a card could have special chips marked as *to_bus* for connecting to the board bus directly (see chip `CanBus` in Fig. 2).

Element is the last decomposition layer of the hardware device. Elements are parts of a chip that provide more detailed functionality. In particular, they are carriers of the ports of the chip. Every chip port provides special functionality for processing input/output, and, precisely, this functionality is encapsulated into the elements. A single element can implement a number of ports.

Hardware elements communicate with each other and the environment via ports. A single hardware element can have several groups of ports of the same type. To represent these groups, we use the **PortGroup** entity. **PortGroup**'s properties are *kind*, *type*, and *multiplicity*. There are two kinds of ports: *internal* (for internal communication of the hardware elements inside of the module), and *panel* (for external communication of the module). The **PortGroup**'s *type* addresses the control of connections for ports that belong to the group. There are special rules for types which can be connected. These roles are expressed in the textual device management DSL. **PortGroup**'s *multiplicity* refers to the number of ports in the particular **PortGroup**. All ports from a **PortGroup** have the same type and kind. Ports of various chips/elements may have the same type. Additionally, the ports of a **PortGroup** are enumerated. The **Port**'s *number* is a significant property, as ports of the same type are often numbered sequentially throughout the entire module.

Essentially, a **PortGroup** is a collection of multiple entry/exit points (ports) of a hardware element with sharing properties. It could be said that **PortGroup**'s properties are a very simple interface of such a point. We do not need explicit interfaces for **PortGroup**/**Port**, as signals and data that are passed through the ports are out of scope of device management. On the other hand, we need not only port groups, but separate ports as well to control port to port connections in our specification: drivers need access to this information to provide network management functions. That is why using layer connections, similarly to ROOM [8], is not suitable in our case.

Some chips could be connected to each other without ports, e.g., voltage sensors `VltPT` are linked like that to the `CanBus` chip (see Fig. 6).

We have omitted a considerable number of parameters of hardware elements, in particular, the complex name/type hierarchy and some additional entities describing it via the means of the device management DSL. The first version of our visual language included all of this information (at the time we did not properly understand the role of a programming domain specific language). We had created a really complicated visual language, and implemented a lot of property sheets for each language construction in a graphical editor. This elicited negative feedback from the users due to the reluctance of the product line developers to use the modeling tool for everyday work, which required them to specify a lot of details in the property sheets.

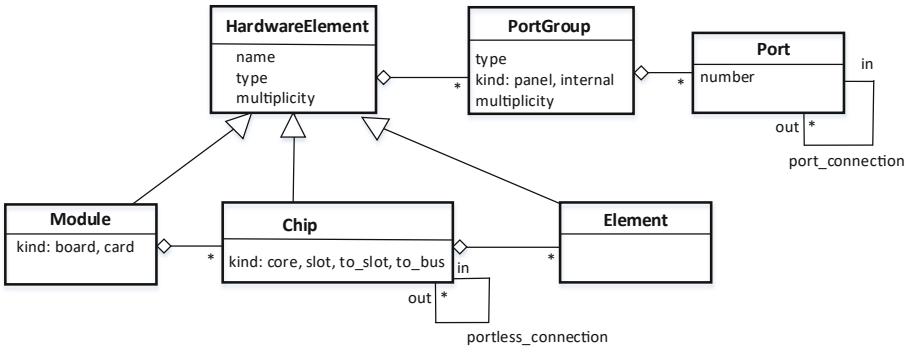

Fig. 1. Language metamodel

4 View Types

Our visual language is intended to provide the ability to browse the specifications on device management DSL to simplify discussing the system structure, renew knowledge about the selected system part (the specification usually has a lot of details, and developers can forget them very quickly), as well as make the introduction of a new employee to the project easier. No new entities can be added using the visual modeling tool, and no editing facilities are provided. We are following the ideas of the visualization toolkit for the Cobol reengineering system described in [6]: visualization of existing software-related entities and their relations using numerous views.

Multiple well-elaborated views are required to enable suitable browsing of a complicated textual specification or program. The multiple view point concept [25] is well-known in the model-based approach: it is used everywhere, from structure analysis approaches to object-oriented notations and UML. However, in our situation, we need to concentrate on this issue due to the following fact. Various views are easy to create because they are not manually constructed, but generated on developer request based on the existing specification. Therefore, we

provide a large number of view types so the developers will not need to spend their efforts on creating them manually. Furthermore, view type variety helps to understand, discuss and analyze the system better.

The visual language we present supports the following view types:

- *Structured view* presents all structural hardware elements of a system to be developed (see Fig. 2); since this is a large volume of information, no connections and elements of chips are depicted. However, chip ports are depicted directly on chips.
- *Composite view* is similar to structured view, but it is augmented with multiplicity indicators in order to depict the instances of chips, elements, and ports (see Fig. 3). This is the reason why this view is not as large as the previous one. To ensure this, in contrast with the previous view type, the elements are shown inside of chips. Furthermore, port connections are depicted as well. However, using multiplicity augmentations does not allow to depict port to port connections because the view presents grouped ports only.
- *Datapath view* presents the data flow process which indicates the main functionality of the module (see Fig. 4). In this view, all other details are omitted.
- *Control view* presents chips, ports, and connections of the module which control the main functionality (see Fig. 5). In short, chips and elements from the previous view perform tasks, and hardware elements from this view control them.
- *Service view* presents the service functionality of the module, e.g., control of temperature or voltage sensors, fan controllers (see Fig. 6).

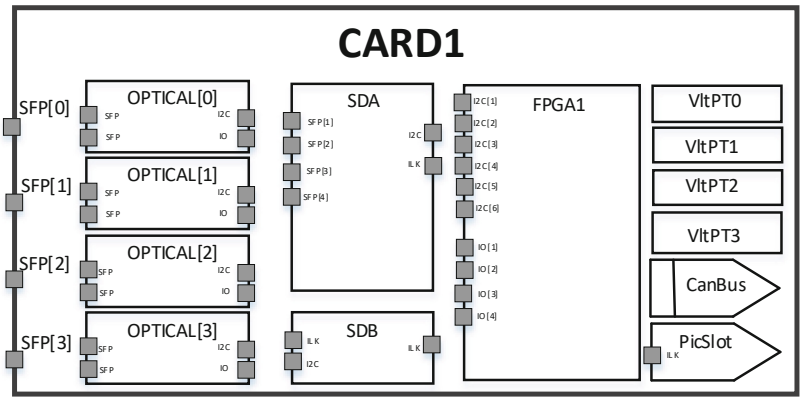

Fig. 2. Example of Structured view

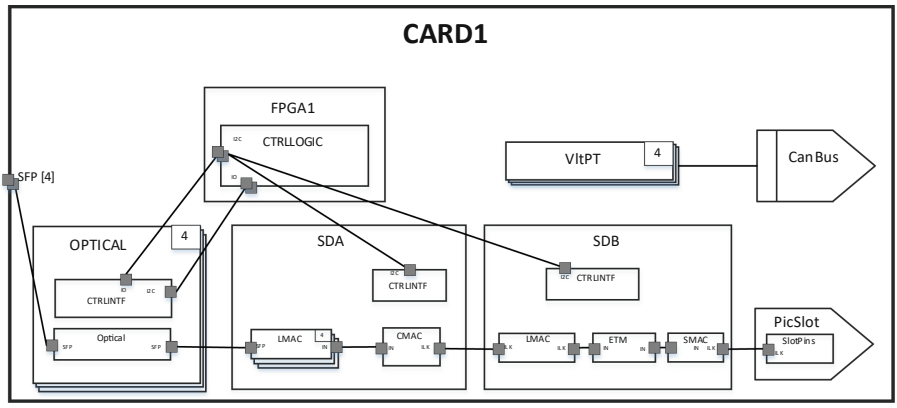

Fig. 3. Example of Composite view

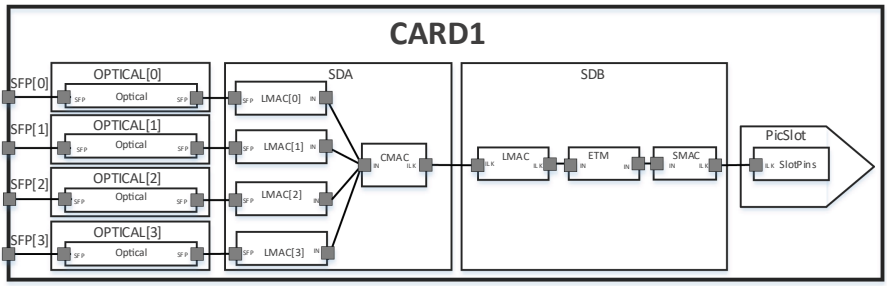

Fig. 4. Example of Datapath view

5 Tool

We have implemented a programming prototype for our visual language using the Eclipse Modeling Tools Xtext/EMF/Sirius [13,31]. We have specified the grammar of our device management DSL using EBNF (Extended Backus-Naur Form). We put some additional information into the grammar to convert it properly into Ecore (i.e., a visual language metamodel), indicating which constructs can be grouped into metaclasses, and which grammar relations map into references or inheritances. Next, using Sirius, we map the Ecore metamodel into the graphical notation and generate the graphical editor.

It should be noted that Eclipse Modeling Tools allows to develop a target visual tool and an environment for textual DSL that are seamlessly connected to each other. In our case, seamlessness refers to the automatic synchronization of constructions on diagrams and in the textual editor. Moreover, Eclipse Modeling Tools provides rapid development process, which is very important taking into account the iterative process of development based on regular user feedback. However, implementing non-standard features using Sirius can be problematic.

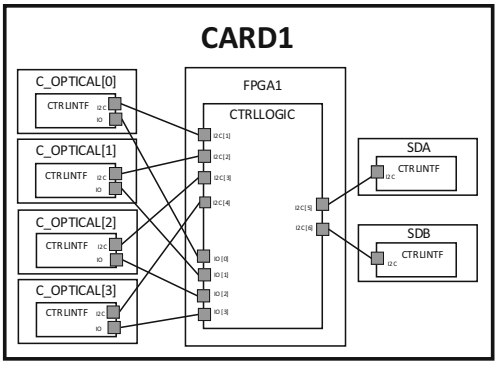

Fig. 5. Example of Control view

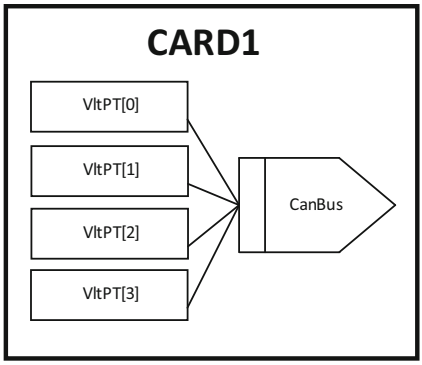

Fig. 6. Example of Service view

In fact, there is a wide variety of aspects that should be kept in mind while choosing a development technology for an industrial solution.

To validate our target tool, we have implemented 15 modules and demonstrated results and tool features to the product line developers. They have given us positive feedback, but insisted on *Structured view* being able to show all of important structural elements of a module on a single diagram, ignoring the big size of such a diagram for real life modules. Following [21], we would like to not ignore the use process, even if we imagined it in a different way (i.e., we thought that the *Composite view* would be enough for a high-level view). However, we will have more feedback after a full introduction of our tool into the development process.

6 Related Work

Hierarchical component models are widely used for modeling real-time systems starting from SDL (70's–90's) [1]. In ROOM [8], the concept of components with ports and interfaces was thoroughly elaborated on, including various kinds of ports, layer connections, and deep integration ports/interfaces with an extended final state machine. Furthermore, this model was used in UML in composite structured diagrams [3], in the real-time UML profile MARTE [2], and SysML [5]. In avionics the special modeling language AADL is used, including features for modeling hardware elements [15].

We have used components and ports from ROOM/MARTE in our visual language, allowing three decomposition levels (module, chip, element). However, in contrast to ROOM, our components are not black boxes: elements placed on one chip can be connected directly to other elements in another chip via element ports. We also allow to create simple relations between chips without ports (in Service view). At last, we do not use a final state machine and port

interfaces (definition data and messages which can be transferred via ports that include this interface). A final state machine is not suitable for hardware drivers, and events (alarms and commands) which drivers process are not connected to hardware ports and are defined separately.

In fact, we have used the subset of ROOM/MARTE that was suitable for our domain, properly changing some details. We have employed the Domain Specific Modeling (DSM) paradigm [20]: instead of customising existing modeling standards, we utilized all their features we needed to create our own small visual language. Using a standard is justified if one is able to use existing tools which support the standard, saving effort on tool development. However, the extension mechanism of modeling standards is not flexible enough, and customization capabilities of modeling tools are quite limited. On the other hand, DSM tools (especially Eclipse Modeling Tools) are mature nowadays. Moreover, it was necessary for us to integrate visual language support with the DSL programming toolset. Consequently, the DSM approach is more efficient in our case than using standard visual modeling languages/tools.

The concept of a product line was actively researched and introduced to the software industry in 1990–2000 [7]. The idea behind it is developing and promoting a set of products together for a specific market segment, and, at the same time, building the products from common (reusable) assets. The significant number of methods and tools were created to support software product lines. Below some recent reviews can be found. In [10], a systematic literature review of the empirical studies on software product lines is presented (reviewed studies were published between 2000 and 2018). In [14], variability metrics for software product lines are collected and analyzed. Approaches to product line evolution are reviewed in [27]. A survey of software product line management tools is presented in [29]. Actually, the topic is a matter of practice rather than intensive research nowadays.

Domain specific programming languages and model-based approach are used in product line development. In [17], the product line is considered as a software factory based on development by assembly, i.e. a significant part of the target applications is supplied from ready-built and built-to order components. However, it is more of a conceptual framework than a set of real development assets (tools, languages, models, etc.). It should be also stressed that it does not support any telecommunication specifics.

The model-driven approach is actively being employed in the context of variability management in product lines [9,18,33,35]. There is a special branch of this approach that focuses on feature modeling [19,34,36], including development of various artifacts, e.g., documentation [24]. The problem of these approaches is the high complexity of variability languages and procedures. Additionally, it requires a formal specification of the reusable artifacts.

A lot of research is dedicated to generation of applications in a product line based on model-based specifications [12,17,22]. The problem of these approaches is creating a labor-intensive and inflexible development infrastructure, which becomes problematic in case of market changes [23].

It should be noted that a large number of model-based approaches has already been created. However, there is a problem with its wide adoption in the industry [30]. Product lines appear to be a suitable context for model-based development combined with domain-specific modeling and programming languages [17,20]. Thus, both research and industrial communities need reports on successful large-scale applications of the model-based approach in order to exchange their successful and unsuccessful experiences.

Finally, let us note that software/hardware telecommunication product lines have become more and more labor-intensive, involving hundreds and even thousands of developers. Meanwhile, there is a lack of research for telecommunication product lines: we have found [11] concerning performance variability, and [28] that considers the problem of defining a product line's scope. In particular, we were not able to find research on the device management problem.

7 Conclusion

In this paper, we have proposed a domain specific visual language for device management in a telecommunication product line. The language addresses the need for browsing hardware specifications using the multiview concept. The implementation of the language in Eclipse Modeling Tools is presented.

The visual language proposed in the paper, could be extended, providing view on modules (boards and cards) of product line. The most obvious relationship between board and card is "a card could be inserted into a board". However, we plan to move ahead basing on feedback and demands of device management engineers. It means we will develop features, which are most required.

As a further research direction, we plan to apply the concept of view-to-view transformation [23] to produce more views demanded by the various needs of the developers. One of the main tasks to solve in order to achieve this goal is the layout problem: the generated views should be arranged properly. One of the partial solutions of this problem is allowing manual layouting. In any case, if the automatic layouting is not good enough, the tool will not be accepted by users.

Another potential research direction is the support of planned reuse and variability management of device management specifications. We could extend the visual language by feature modeling formalism [19,36] for visual variability management. It would be the next step after visualization of hardware structure for device management.

Finally, using knowledge graphs and ontology engineering [16,26] as an alternative approach to gather and access information required for device management appears to be promising. This approach may be efficient due to the significant heterogeneity of the device management information, although we have not highlighted this issue in the paper.

References

1. ITU Recommendation Z.100: Specification and Description Language. ITU-T (2002)
2. A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems. OMG, June 2008
3. Unified Modeling Language (UML). OMG (2013)
4. Meta Object Facility (MOF) Core Specification. OMG (2015)
5. OMG Systems Modeling Language. OMG, November 2019
6. Baburin, D.E., Bulyonkov, M.A., Emelianov, P.G., Filatkina, N.N.: Visualization facilities in program reengineering. Program. Comput. Softw. **27**(2), 69–77 (2001)
7. Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Heidelberg (2010). https://doi.org/10.1007/3-540-28901-1
8. Selic, B., Gullekson, G., Ward, P.T.: Real-Time Object-Oriented Modeling. Wiley, New York (1994)
9. Buchmann, T., Greiner, S.: Managing variability in models and derived artefacts in model-driven software product lines. In: Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2018, Funchal, Madeira, Portugal, 22–24 January 2018, pp. 326–335. SciTePress (2018)
10. Chacón-Luna, A.E., Gutiérrez, A.M., Galindo, J.A., Benavides, D.: Empirical software product line engineering: a systematic literature review. Inf. Softw. Technol. **128**, 106389 (2020)
11. Cvetković, R., Nešković, S.: An approach to defining scope in software product lines for the telecommunication domain. In: Catania, B., Ivanović, M., Thalheim, B. (eds.) ADBIS 2010. LNCS, vol. 6295, pp. 555–558. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15576-5_44
12. Dageförde, J.C., Reischmann, T., Majchrzak, T.A., Ernsting, J.: Generating app product lines in a model-driven cross-platform development approach. In: 49th Hawaii International Conference on System Sciences, HICSS 2016, Koloa, HI, USA, 5–8 January 2016, pp. 5803–5812. IEEE Computer Society (2016)
13. Eclipse Project: Eclipse Sirius. https://www.eclipse.org/sirius/
14. El-Sharkawy, S., Yamagishi-Eichler, N., Schmid, K.: Metrics for analyzing variability and its implementation in software product lines: a systematic literature review. In: Berger, T., et al. (eds.) Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, vol. A, p. 33:1. ACM (2019)
15. Feiler, P.H., Gluch, D.P.: Model-Based Engineering with AADL - An Introduction to the SAE Architecture Analysis and Design Language. SEI Series in Software Engineering, Addison-Wesley, Boston (2012)
16. Gavrilova, T.: Ontological engineering for practical knowledge work. In: Apolloni, B., Howlett, R.J., Jain, L. (eds.) KES 2007. LNCS (LNAI), vol. 4693, pp. 1154–1161. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74827-4_144
17. Greenfield, J., Short, K., Cook, S., Kent, S., Crupi, J.: Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. Wiley, Chichester (2004)
18. He, X., Fu, Y., Sun, C., Ma, Z., Shao, W.: Towards model-driven variability-based flexible service compositions. In: 39th IEEE Annual Computer Software and Applications Conference, COMPSAC 2015, Taichung, Taiwan, 1–5 July 2015, vol. 2, pp. 298–303. IEEE Computer Society (2015)

19. Kang, K.C., Lee, J., Donohoe, P.: Feature-oriented product line engineering. IEEE Softw. **19**(4), 58–65 (2002)
20. Kelly, S., Tolvanen, J.P.: Domain-Specific Modeling: Enabling Full Code Generation. Wiley, New York (2008)
21. Kelly, S., Pohjonen, R.: Worst practices for domain-specific modeling. IEEE Softw. **26**(4), 22–29 (2009)
22. Kim, S.D., Min, H.G., Her, J.S., Chang, S.H.: DREAM: a practical product line engineering using model driven architecture. In: Third International Conference on Information Technology and Applications (ICITA 2005), 4–7 July 2005, Sydney, Australia, pp. 70–75. IEEE Computer Society (2005)
23. Koznov, D.V.: Process model of DSM solution development and evolution for small and medium-sized software companies. In: Workshops Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference, EDOCW 2011, Helsinki, Finland, 29 August–2 September 2011, pp. 85–92. IEEE Computer Society (2011)
24. Koznov, D.V., Romanovsky, K.Y.: DocLine: a method for software product lines documentation development. Program. Comput. Softw. **34**(4), 216–224 (2008)
25. Kruchten, P.: The 4+1 view model of architecture. IEEE Softw. **12**(6), 42–50 (1995)
26. Kudryavtsev, D., Gavrilova, T.: Diagrammatic knowledge modeling for managers - ontology-based approach. In: KEOD 2011 - Proceedings of the International Conference on Knowledge Engineering and Ontology Development, Paris, France, 26–29 October 2011, pp. 386–389. SciTePress (2011)
27. Marques, M., Simmonds, J., Rossel, P.O., Bastarrica, M.C.: Software product line evolution: a systematic literature review. Inf. Softw. Technol. **105**, 190–208 (2019)
28. Myllärniemi, V., Savolainen, J., Männistö, T.: Performance variability in software product lines: a case study in the telecommunication domain. In: 17th International Software Product Line Conference, SPLC 2013, Tokyo, Japan, 26–30 August 2013, pp. 32–41. ACM (2013)
29. Pereira, J.A., Constantino, K., Figueiredo, E.: A systematic literature review of software product line management tools. In: Schaefer, I., Stamelos, I. (eds.) ICSR 2015. LNCS, vol. 8919, pp. 73–89. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-14130-5_6
30. Petre, M.: UML in practice. In: 35th International Conference on Software Engineering, ICSE 2013, San Francisco, CA, USA, 18–26 May 2013, pp. 722–731. IEEE Computer Society (2013)
31. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework. Addison-Wesley Professional, Boston (2008)
32. Tolvanen, J., Kelly, S.: Applying domain-specific languages in evolving product lines. In: Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, vol. B, Paris, France, 9–13 September 2019, pp. 65:1–65:2. ACM (2019)
33. Tolvanen, J., Kelly, S.: How domain-specific modeling languages address variability in product line development: investigation of 23 cases. In: Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, vol. A, Paris, France, 9–13 September 2019, pp. 24:1–24:9. ACM (2019)
34. Usman, M., Iqbal, M.Z., Khan, M.U.: A product-line model-driven engineering approach for generating feature-based mobile applications. J. Syst. Softw. **123**, 1–32 (2017)

35. Verdier, F., Seriai, A.-D., Tiam, R.T.: Combining model-driven architecture and software product line engineering: reuse of platform-specific assets. In: Hammoudi, S., Pires, L.F., Selic, B. (eds.) MODELSWARD 2018. CCIS, vol. 991, pp. 430–454. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-11030-7_19
36. Yang, G., Zhang, Y.: A feature-oriented modeling approach for embedded product line engineering. In: 12th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2015, Zhangjiajie, China, 15–17 August 2015, pp. 1607–1612. IEEE (2015)