Experience from Implementing a Complex Eclipse Extension for Software Product Line Engineering

Sascha El-Sharkawy[1], Christian Kröher[1], Holger Eichelberger[1], Klaus Schmid[1]

[1]Software Systems Engineering, Institute of Computer Science,
University of Hildesheim, Germany
{elscha, kroeher, eichelberger, schmid}@sse.uni-hildesheim.de

Please cite this publication as follows:
Sascha El-Sharkawy et al. "Experience from Implementing a Complex Eclipse Extension for Software Product Line Engineering". In: *Proceedings of the on Eclipse Technology eXchange (ETX'15)*. ACM, 2015, pp. 13–18. DOI: 10.1145/2846650.2846654.

The corresponding BibTeX-entry is:

```
@INPROCEEDINGS{El-SharkawyKroeherEichelberger+15,
  author = {Sascha El-Sharkawy and Christian Kr{\"o}her and Holger Eichelberger and Klaus Schmid},
  title  = {Experience from Implementing a Complex Eclipse Extension
            for Software Product Line Engineering},
  booktitle={Proceedings of the on Eclipse Technology eXchange (ETX'15)},
  publisher = {ACM},
  year   = {2015},
  pages = {13--18},
  doi    = {10.1145/2846650.2846654}
}
```

Experience from Implementing a Complex Eclipse Extension for Software Product Line Engineering

Sascha El-Sharkawy, Christian Kröher, Holger Eichelberger, Klaus Schmid
University of Hildesheim, Universitätsplatz 1
31141 Hildesheim, Germany
+49 5121 883 {40336, 40340, 40334, 40332}
{elscha, kroeher, eichelberger, schmid}@sse.uni-hildesheim.de

Abstract
Software Product Line Engineering (SPLE) is a systematic approach for the development of related software products. These products share a common infrastructure but vary with respect to their individual capabilities, called variabilities. Variability management is a key part of SPLE and is responsible for developing, combining and configuring such variabilities. As these activities are inherently complex, SPLE significantly benefits from tool-support.

We developed a customizable Eclipse extension for SPLE that consists of around 38 plug-ins. The resulting tool, called EASy-Producer, extends the Eclipse IDE by the capability to support the creation and management of software product line projects. To provide this capability, EASy-Producer utilizes the extension concepts of the Eclipse platform and integrates additional frameworks, like Xtext.

In this paper, we share our experience while applying the Eclipse technologies and, in particular, realizing specific capabilities of our tool using the Eclipse framework. The focus of this paper is on our lessons learned regarding managing workspace information and conflicting build mechanism as well as using Eclipse extensions outside of Eclipse. These lessons serve as an input to the Eclipse community and may help other developers in realizing a complex Eclipse extension.

Categories and Subject Descriptors
D.2.13 [**Reusable Software**]: *Domain engineering*; D.2.6 [**Programming Environments**]

Keywords
EASy-Producer, Software Product Lines, Eclipse.

ETX'15, October 27, 2015, Pittsburgh, PA, USA
ACM. 978-1-4503-3904-9/15/10...
http://dx.doi.org/10.1145/2846650.2846654

1. Introduction
Successful software typically evolves into a set of variants (a *software family*). Either variants address different market segments, like for different platforms, high-end vs. low-end systems, etc. or they may even be created for individual customers, with potentially each customer requiring specific features. In the most extreme case, this can amount to more than 100 variants of a basic system type that an organization may need to develop per year. In such a situation copying an existing solution and adapting it to specific customer demands may seem to be a quick and easy approach. However, the more variants are developed, the more complex consistent management and maintenance of all products becomes.

Software Product Line Engineering (SPLE) is one industrial-ready technique to create, manage, and maintain such software families [6]. This technique is heavily researched since the early 1990s and has been applied in many industrial projects. Its core idea is to develop software as a set of related products that share a common infrastructure (*commonalities*) but vary with respect to their individual capabilities (*variabilities*). This variation is at the heart of SPLE: developing generic artifacts that can be configured and combined to create a wide range of specific product variants.

Variability Management (VM) is a key discipline in SPLE that is responsible for developing and managing the variabilities of a Software Product Line (SPL). VM consists of four main VM Activities (VMA) [6]:

VMA 1: Modeling variability
VMA 2: Modeling the relation between variability and generic artifacts
VMA 3: Describing the configuration of an instance
VMA 4: Deriving customized products

In general, performing these activities consistently is a rather complex task, which becomes even more complex the more variabilities have to be managed. Thus, SPLE benefits significantly from tool-support [6].

The Eclipse platform provides an extensible infrastructure that allows the integration of custom plug-ins to extend the IDE by further capabilities. A major advantage of this infrastructure is that extensions can reuse already available plug-ins. This reduces the effort for creating individual editors, integrating version control, etc.

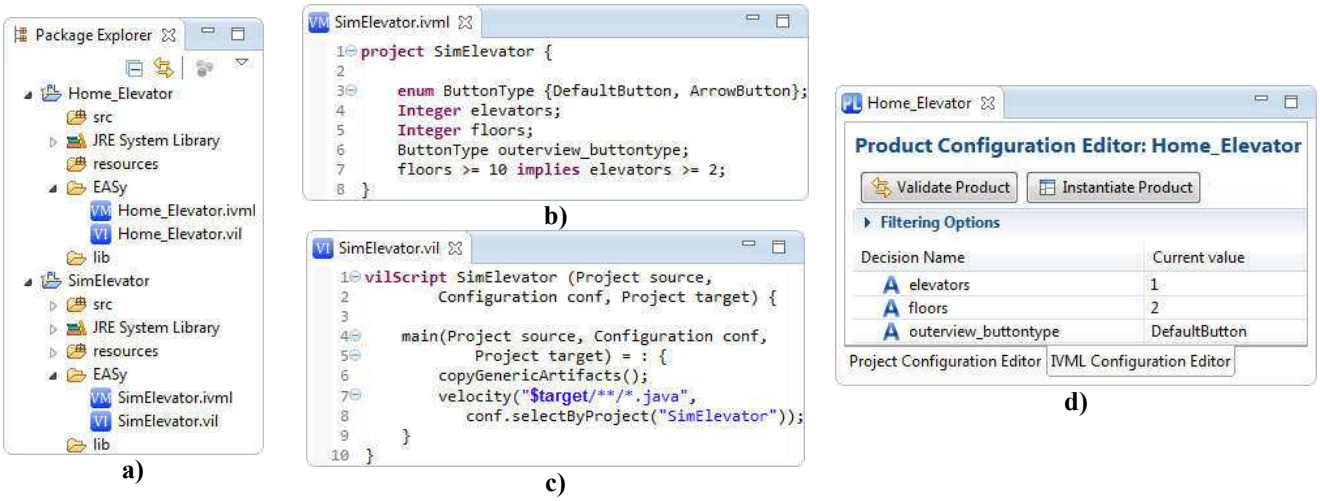

Figure 1: SimElevator SPL in EASy-Producer

In this paper, we share our experience from developing EASy-Producer[1], a tool for providing product line support in Eclipse. This tool seamlessly integrates with the general Eclipse IDE. For example, projects for developing product lines do not fundamentally differ from usual Eclipse projects and, thus, provide the same capabilities for project management. Further, the tool introduces customizable product line capabilities to Eclipse, leveraging different extension mechanisms provided by the Eclipse platform. In contrast to other publications on EASy-Producer like [1], the focus of this paper is on the technical realization of this seamless integration and extensions. In particular, we share our experiences from realizing a complex Eclipse extension in terms of lessons learned. These lessons may help developers in realizing extensions for and with the Eclipse platform.

This paper is structured as follows: In Section 2 we introduce EASy-Producer as part of the Eclipse IDE and describe its basic application by an example SPL. Understanding the application of the tool lays the foundation for discussing the required implementation in Section 3. In particular, we discuss the realization of those parts that enable the support for the four VMAs in Eclipse. Based on these implementation details, we provide our lessons learned as the main contribution of this paper in Section 4. In Section 5, we discuss related work and conclude in Section 6.

2. EASy-Producer

In this section, we introduce EASy-Producer as an extension for SPL development with Eclipse. We discuss the support along the four VMAs introduced in Section 1 and an illustrative example: the *SimElevator* SPL. This SPL provides different variants of an elevator simulator implemented in Java, where Velocity mark-ups[2] are used to realize the variabilities. At the end of this section, we discuss the support for advanced SPL techniques.

EASy-Producer extends the Eclipse IDE by a specific project nature: the "EASy-Producer Nature", which can be added to any project in Eclipse tagging it as a Product Line Project (PLP). A PLP is a combination of a product line infrastructure and derived product and, thus, does not fundamentally distinguish between those SPL instances. Figure 1 a) shows the *SimElevator* PLP. A PLP always contains an *EASy*-folder, which by default contains the following files:

- At least one `*.ivml` file in which we define the variability model of the SPL (cf. VMA 1). For the *SimElevator* SPL, we define the number of elevators and floors, and the button type as variabilities (Figure 1b)) using the IVML language[3]. Further, we define that ten or more floors require at least two elevators (Figure 1b), line 7) as a constraint.
- At least one `*.vil` file in which we define the relation between the variabilities and the implementation (cf. VMA 2) and the configuration of artifacts (cf. VMA 3) using the VIL language[4]. For the *SimElevator* SPL, we want to replace the Velocity mark-ups with the values of the variabilities defined in the variability model (Figure 1c), lines 7 and 8).

Further, EASy-Producer supports variability-aware artifact generation using `*.vtl` files (see the VIL language specification[3] for more information). All three languages (IVML, VIL, VTL) are custom-developed DSLs.

To derive a configured product (cf. VMA 4), EASy-Producer provides a product line editor in which we define a new product line member for the *SimElevator* PLP. We call this member *Home_Elevator*. The tool then automatically creates a new PLP with this name, which holds certain references to the parent product line project *SimElevator* PLP. These references allow importing the information defined in the parent PLP's `*.ivml` and `*.vil` files. Thus, the product line editor for the *Home_Elevator* displays the variabilities defined in the *SimElevator* PLP. Figure 1d) shows this editor

[1] EASy stands for Engineering Adapative Systems
[2] The Apache Velocity Project: http://velocity.apache.org/
[3] IVML Language Specification: http://projects.sse.uni-hildesheim.de/easy/docs/ivml_spec.pdf
[4] VIL Language Specification: http://projects.sse.uni-hildesheim.de/easy/docs/vil_spec.pdf

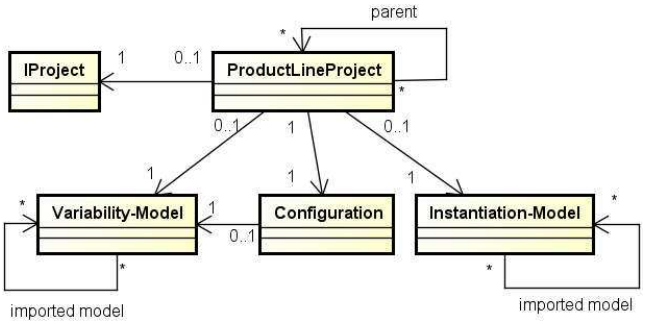

Figure 2: Simplified project structure.

for the *Home_Elevator* in which we already defined the number of elevators and floors as well as the button type for that product variant.

This configuration is input to the *Home-Elevator*'s instantiation process, which is started by pressing the *Instantiate Product* button (cf. Figure 1d)). EASy-Producer then automatically checks if any constraints of the variability model are violated using its custom reasoner, a tool for checking the configured values of variabilities against the defined constraints. If no constraint is violated, which is the case in the *Home_Elevator* configuration, the tool performs the instantiation process defined by the content of the `*.vil` file. The final result is the presence of the *SimElevator* (re-) sources in the respective folders of the *Home_Elevator* PLP. In these artifacts all mark-ups are resolved based on the given configuration.

The PLP and referencing concepts are key enablers to support even more complex SPL settings:

- *Staged Configuration:* Assuming that the number of elevators for all *Home_Elevator* variants should always be 1, we configure only this variability (*elevators*) in that PLP. Specific variants derived from that PLP will now only allow the configuration of the number of floors and the button type as the number of elevators is already defined in the previous stage.
- *Partial Instantiation:* As the number of elevators for *Home_Elevator* variants is now constantly 1, also the related artifacts can already be instantiated at this stage. All other artifacts will be instantiated as part of the instantiation of a specific variant derived from the *Home_Elevator* PLP.
- *Multi Software Product Lines (MSPL):* In addition to the *SimElevator* SPL, we may have another SPL in place that offers further functionalities we want to integrate into specific product variants. In this case, EASy-Producer allows defining additional parent PLPs for an already derived PLP, e.g. the *Home_Elevator*. This also adds new references to the additional parent PLP, which makes the variabilities defined in the PLP also available in *Home_Elevator*.

In EASy-Producer, these settings can be arbitrarily combined, as the tool does not fundamentally differentiate between product lines and products; both are PLPs. We describe this in more detail in [1]. While in [1] we focused on

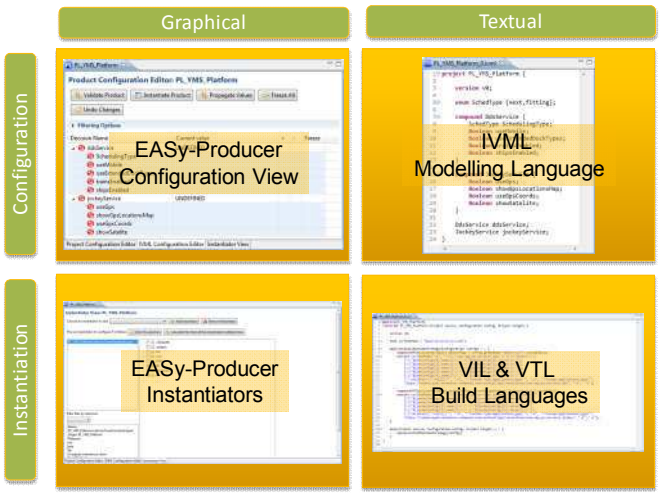

Figure 3: Product line modeling in EASy-Producer.

the product line concepts supported by and realized in the EASy-Producer toolset, we focus here on experiences and lessons learned from realizing these concepts in the Eclipse-infrastructure.

3. Realization

We will now present the key concepts of the realization of EASy-Producer. In the first subsection, we show how we implemented the project structures as described in Section 2. In Section 3.2, we present the extendable architecture of EASy-Producer and show how different extension concepts, provided by Eclipse, are used to realize the tool.

3.1 SPL Development Support

In Section 2, we mentioned that EASy-Producer facilitates an arbitrary linkage of Eclipse projects (actually, the relations among all projects can be an arbitrary, acyclic graph). Each project can contribute to the instantiation process of other projects or make use of other projects during its instantiation process (cf. VMA 4 in Section 1).

This is realized as shown in Figure 2. Each PLP is managed by an instance of `ProductLineProject`, which is associated with an `IProject`, Eclipse's data objects for managing projects of its workspace. We use the provided project capabilities of Eclipse to be able to use existing project management tools, e.g., for versioning.

These `ProductLineProjects` know their parent projects, which are needed for their instantiation process. These `ProductLineProjects` also contain the definition of the variability model (see Section 2, `*.ivml` files) and the instantiation process (see Section 2, `*.vil` and `*.vtl` files). These models may be composed of other models of the same type. Further, the `ProductLineProjects` contain a `Configuration`, which stores temporary user input as well as propagated values of the reasoner.

Even if the `ProductLineProjects` reference their parent projects, they need not to be physically available in the workspace. This enables collaborative development, where not all developers of a product should be able to access the

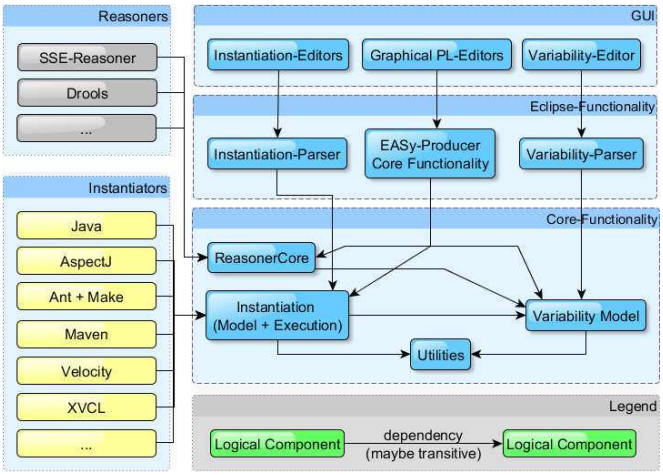

Figure 4: Simplified architecture of EASy-Producer.

code of other projects. Only for the instantiation process, supplying projects must be locally available.

3.2 Expandable Architecture

EASy-Producer follows the traditional distinction of variability modeling and artifact instantiation as shown in Figure 3. This is enhanced by mechanisms for product line composition and specialization to provide specific support for variability-rich software systems. In addition, EASy-Producer provides its support for all VMAs in two ways (Figure 3): *simplified graphical views*, which are mostly intended for non-expert users, and *textual modeling views* (using the IVML, VIL, and VTL languages), which unleash the full power of EASy-Producer. The later ones are mostly intended for expert users.

Furthermore, EASy-Producer can be customized to fulfill the requirements of different usage scenarios. This is realized by the architecture shown in Figure 4 and the capabilities provided by Eclipse. We summarized plug-ins, which belong together, to reduce the complexity of the implementation and to show only logical components.

The *Utilities* provide functionalities, which are useful for most of the other bundles, e.g., a check whether EASy-Producer runs inside Eclipse or is only loaded as a Java library, or common super classes for the different data models. The data management is split into two models: the *Variability-Model* and the *Instantiation-Model* for managing the instantiation process. The expressive *Variability-Model* was designed to be able to handle very complex modeling scenarios from industry [5] up to complex topologies [2, 9] (cf. VMA 1 in Section 1). Also the *Instantiation-Model* was designed as an expressive model, but should also be extensible to support individual instantiation mechanisms for certain artifact types if needed (cf. VMA 2 in Section 1). These models are defined in a textual language, which are written in textual editors and interpreted by parsers generated by Xtext (*Instantation-/Variability-Parsers/Editors*). We decided to keep the variability model and the instantiation model independent of Xtext and use the generated Ecore models only for creating and updating those models. This allows to exchange the human readable text files by alternative model representation, which are not human readable, but can be processed much more efficiently, e.g., as part of an automated high performance setup. The *ReasonerCore* bundle allows to register different reasoner implementations to validate a (partial) configuration against the definition of the variability model. The *EASy-Producer Core Functionality* component integrates the different components. Finally, the *Graphical PL-Editors* bundle provides simplified graphical editors (see Figure 3, graphical editors), which serves as an alternative to the more powerful textual editors generated by Xtext.

For the extensible realization of our architecture, we use different extension techniques provided by Eclipse to address different needs of the tool:

- *Plugin.xml + Manifest.mf:* These are the usual manifest files to make contributions to Eclipse. We use extension points defined by Eclipse to register editors and wizards, make contributions to existing menus, extend the existing property pages of Eclipse, or to adapt our editor to fulfill customer needs.
  - *Activator:* A *Manifest.mf* of a plug-in may specify an Activator to control the plug-in's life-cycle. Its startup method is executed when the first class of the plug-in is loaded and executed. However, its execution is not guaranteed.
    For instance, we use this technique to register a listener to Eclipse's workspace to monitor relevant project changes. This is part of the *EASy-Producer Core Functionality*. Classes of this bundle will be executed at least if the user starts working with EASy-Producer, which forces the execution of the Activator.
  - *IStartup:* The UI-packages of Eclipse offer this extension point to execute code after the Eclipse workbench was loaded[5]. We use this mechanism to scan the workspace for PLPs, directly after the UI was loaded by Eclipse, as we need to know which of the related projects are actually available in the workspace.
- *OSGi Declarative Services (DS):* The aforementioned techniques are not sufficient for each aspect of EASy-Producer. Especially the different reasoner implementations or the instantiation extensions should be registered with the related plug-ins, without any dependencies to the UI. For this, the use of Activators or the *IStartup* extension was not an option. Hence, we decided to use the OSGi Declarative Services for a guaranteed registration of the extensions at the related bundles during the startup of Eclipse.

[5] The Eclipse Foundation. Startup: http://help.eclipse.org/mars/topic/org.eclipse.platform.doc.isv/reference/extension-points/org_eclipse_ui_startup.html

- *ExtensionRegistry:* At the ScaleLog project[6], an industrial partner needed an extension of a menu provided by a third party plug-in, which did not offer an extension point. For this, we retrieved the `ExtensionRegistry` from the `Platform` and identified the relevant extension. We exchanged the relevant extension via reflection to call first the original action and then our extension if the related menu entry was clicked.

This shows that depending on the specific circumstances different approaches to extension need to be used. In particular, not all needed approaches are directly supported through Eclipse extension mechanisms.

4. Lessons Learned

In this section, we present lessons learned from creating the EASy-Producer product line extension for Eclipse.

4.1 Plug-in State

Eclipse offers mechanisms to save workspace-related information persistently to disk. For instance, the class `org.eclipse.core.runtime.Platform` offers the method `getStateLocation` to create a plug-in related configuration file inside the `.metadata` folder of the workspace. This information is saved independently of any Eclipse projects.

In Section 3.1, we mentioned the complex linkage of PLPs, which is necessary to support arbitrary instantiation processes connecting code from multiple projects. An early prototype of EASy-Producer used the technique from above to maintain the general available PLPs. While creating a new PLP, we added information about these projects to a configuration file inside the `.metadata` folder. This information was used for linking the projects, even if a project was renamed in the meanwhile.

It turned out that this concept leads to problems when projects where imported into the workspace or were temporarily closed and re-opened at a later time. Recent versions of EASy-Producer scan the workspace for available PLPs at startup and use a listener registered with the workspace to get informed when projects are imported or closed, instead of using the configuration file to cope with the aforementioned problem.

4.2 Conflicting Build Mechanisms

EASy-Producer offers its own user-triggered instantiation process. We decided against the usual automatic build process offered by Eclipse, because complex instantiation settings may need several seconds per build. For instance, the instantiation of the QualiMaster[7] project, which involves a consistency check of the variability model and includes a maven build, takes up to 40 seconds. Also the user should be able to control when the build is started, as in complex build-scenarios Eclipse cannot be aware whether all supplying projects are in a consistent state. Furthermore, the classic build capabilities should still be available to support the development of product line artifacts in a traditional way, as known by typical software developers.

This combination of the traditional automatic builders of Eclipse together with the user-triggered instantiation process of EASy-Producer may lead to a conflicting build process. This is the case if EASy-Producer should build files into the project's source and binary folders with a different compiler than configured for the automatic build, e.g., using the AspectJ instantiator of EASy-Producer without having the AspectJ Tools installed in Eclipse. In this case, the Java compiler of Eclipse ignores the aspects and overwrites the generated files of EASy-Producer.

This problem can be avoided by installing the AspectJ tools and add its nature to the project. However, this is not always desired. In such cases, it is possible to build the binary files into another folder than specified by Eclipse. This solves the problem, but Eclipse is no longer able to execute the generated files. This issue can be handled by defining a "run configuration", which instructs Eclipse to execute the compiled files outside of the project's binary folder.

4.3 Eclipse Runtime Check

Some of EASy-Producer's OSGi-bundles were developed to be used outside of Eclipse as plain Java libraries, too. In doing so, some Eclipse components are not loaded even if related libraries are added to the classpath, e.g., the workspace will not be available. Calling such components will throw many exceptions.

Components, which are used both as OSGi bundles and as traditional Java libraries, check whether they run inside Eclipse before they call such operations. This is done by checking whether the following Eclipse properties are defined (not being `null`):

- `eclipse.product`
- `eclipse.home.location`
- `eclipse.pde.launch`

However, this solution requires multiple if-statements in different bundles. A more elegant solution is the application of variability implementation techniques to make those parts configurable as needed by the environment. We plan this for future releases of the tool.

5. Related Work

Different SPL-tools exist that support various aspects of SPL development. In this section, we discuss only those tools that are also available as Eclipse plug-ins or are based on similar technologies as EASy-Producer.

The Dopler tool [3] supports all four VMAs. The tool is developed as a SPL providing an explicit variability model for configuring the tool to a specific domain, e.g. by selecting Eclipse plug-ins and features and even manipulating the behavior of certain plug-ins by parameter settings. While EASy-Producer does not support the customization of

[6] ScaleLog project: https://sse.uni-hildesheim.de/en/ScaleLog
[7] QualiMaster project: http://www.qualimaster.eu

individual plug-ins, selecting desired features and plug-ins is possible. Further, we assume that, similar to our tool, Dopler extensions facilitate the Eclipse plug-in-mechanism. However, a detailed description on the technical realization of Dopler extensions is not publicly available.

FeatureIDE [12] and pure::variants [7] also support all VMAs and can be configured by selecting desired plug-ins and features. Further, both tools can be extended by custom Eclipse plug-ins using the extension points provided by the respective tool, e.g. to provide extensions to the feature model editor [12] or custom transformation modules [8]. This is similar to EASy-Producer, however, our tool also supports further extension mechanisms (cf. Section 3.2).

The FaMa framework is available as OSGi bundles [10]. The focus of this tool is exclusively on VMA 1: modeling and analyzing variabilities using a cardinality-based feature modeling approach. New variability meta-models, analysis tools, etc. can be added, which requires the manipulation of configuration files to register new classes, which in turn have to implement predefined interfaces. EASy-Producer instead offers explicit extension points and registries that automatically search for extensions like new model types.

Additional Eclipse-based tools exist, like REMiDEMMI [11] and FeatureMapper [4]. However, we neither found information on possible adaptations or extensions of these tools nor on implementation details.

6. Conclusion

Software Product Line Engineering (SPLE) significantly benefits from adequate tool support. However, realizing specific concepts and capabilities to support classical SPLE as well as more complex settings, like multi software product lines, requires the combination of different approaches and technologies. The Eclipse platform offers a plethora of already existing plug-ins, mechanism, and frameworks that can be reused and combined for such tool support.

In this paper, we discussed the technical realization of EASy-Producer, an Eclipse extension realizing our research concepts for efficient software product line development. Based on our experience in implementing a complex Eclipse extension, we provided our lessons learned, e.g. regarding conflicts between the traditional build process of Eclipse and custom build processes, or using components both as Eclipse plug-ins and traditional Java libraries outside of Eclipse.

We discussed the technical realization and lessons learned with respect to a specific SPLE tool. However, the presented solutions are not limited to this domain but are generic solutions within the Eclipse framework. Thus, developers may benefit from our experience, e.g. by adopting some of our solutions to their specific problems. As EASy-Producer is licensed under Apache license 2.0. and available on our GITHUB site (*ssehub.net*), interested developers have access to the full implementation details.

Acknowledgments

This work was partially supported by the European Commission in the 7th framework programme through the INDENICA project (grant 257483) and the QualiMaster project (grant 619525). Any opinions expressed herein are solely by the authors and not of the funding agency.

References

[1] Holger Eichelberger, Sascha El-Sharkawy, Christian Kröher, and Klaus Schmid. EASy-Producer – Product Line Development for Variant-Rich Ecosystems. In *Software Product Line Conference (SPLC '14), Volume 2*, pages 133–137, 2014.

[2] Holger Eichelberger and Klaus Schmid. Mapping the design-space of textual variability modeling languages: a refined analysis. *Journal on Software Tools for Technology Transfer*, 1:1–26, 2014.

[3] Paul Grünbacher, Rick Rabiser, Deepak Dhungana, and Martin Lehofer. Model-Based Customization and Deployment of Eclipse-Based Tools: Industrial Experiences. In *International Conference on Automated Software Engineering (ASE '09)*, pages 247–256, 2009.

[4] Florian Heidenreich, Jan Kopcsek, and Christian Wende. FeatureMapper: mapping features to models. In *International Conference on Software Engineering (ICSE '14)*, pages 943–944, 2008.

[5] INDENICA Consortium. Deliverable D2.2.1: Variability Implementation Techniques for Platforms and Services (interim), 2011. http://sse.uni-hildesheim.de/indenica.

[6] Klaus Schmid and Eduardo S. Almeida. Product Line Engineering. *IEEE Software*, 30:24–30, 2013.

[7] pure-systems GmbH. Technical White Paper - Variant Management with pure::variants, 2004. http://www.pure-systems.com/fileadmin/downloads/pv-whitepaper-en-04.pdf.

[8] pure-systems GmbH. pure::variants Extensibility Guide, 2015. Part of pure::variants - SDK (Evaluation).

[9] QualiMaster Consortium. Deliverable D4.1: Quality-aware Processing Pipeline Modeling, 2014. http://qualimaster.eu.

[10] The FaMa Team. FaMa FW - Project Website, 2012. http://www.isa.us.es/fama/?Welcome.

[11] The REMiDEMMI Team. REMiDEMMI - Project Website, 2015. https://sse.uni-due.de/remidemmi.

[12] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. FeatureIDE: An extensible framework for feature-oriented software development. *Science of Computer Programming*, 79:70–85, 2014.