|  |  |  |
| --- | --- | --- |
| Uni_Logo.png |  | Zuschneiden.jpg |
| **Stiftung University of Hildesheim**  Marienburger Platz 22  31141 Hildesheim  Germany |  | **Software Systems Engineering (SSE)**  Institute for Computer Science  Faculty for Mathematics, Natural Science, Economics, and Computer Science |
|  |  |  |



**EASy-Producer**

Engineering Adaptive Systems

**User Guide**

Version 0.4

30.11.2012

**Version**

|  |  |  |
| --- | --- | --- |
| 0.1 | 23.08.2012 | Initial version. |
| 0.2 | 10.09.2012 | Table of content, initial introduction, prerequisites, and installation section added. |
| 0.3 | 22.10.2012 | Changes due to migration to Xtext version 2.3.1, preface added, modification and extension of Sections 1, 3 and 4. Sections 2, 5, 6, and appendix initially added. |
| 0.4 | 30.12.2012 | Section 7 updated. |

**Preface**

EASy-Producer is a Software Product Line Engineering tool developed by the Software Systems Engineering (SSE) group at the University of Hildesheim.

The tool is available as an Eclipse plug-in under the terms of the Eclipse Public License (EPL) Version 1.0

The SSE group hosts the following EASy-Producer update site for easy installation and updates:

<http://projects.sse.uni-hildesheim.de/easy/>

©2012 Software Systems Engineering (SSE) Group, University of Hildesheim, Germany.

Table of Contents

[1. Introduction 6](#_Toc338688603)

[2. Software Product Line Engineering at a Glance 7](#_Toc338688604)

[2.1. Basic Software Product Line Engineering 7](#_Toc338688605)

[2.2. Staged Configuration and Instantiation 7](#_Toc338688606)

[2.3. Multi Software Product Lines 7](#_Toc338688607)

[3. Installation 9](#_Toc338688608)

[3.1. Prerequisites 9](#_Toc338688609)

[3.2. Installation: Step by Step 9](#_Toc338688610)

[3.3. Recommendations 11](#_Toc338688611)

[4. Getting Started: Product Line Engineering is EASy 12](#_Toc338688612)

[4.1. Running Example 12](#_Toc338688613)

[4.2. Defining a New Base Service Platform 13](#_Toc338688614)

[4.2.1. Configuration Space Definition 14](#_Toc338688615)

[4.2.2. Implementation Space Definition 15](#_Toc338688616)

[4.3. Deriving a Domain-Specific Service Platform 17](#_Toc338688617)

[4.3.1. Configuration of a Domain-Specific Service Platform 18](#_Toc338688618)

[4.3.2. Instantiation of a Domain-Specific Service Platform 19](#_Toc338688619)

[5. EASy-Producer in Detail 21](#_Toc338688620)

[5.1. The Product Line Project Structure 21](#_Toc338688621)

[5.2. The Product Line Editor 21](#_Toc338688622)

[5.3. The Project Configuration Editor 21](#_Toc338688623)

[5.4. The IVML Configuration Editor 21](#_Toc338688624)

[5.4.1. Default Reasoning Capabilities 21](#_Toc338688625)

[5.4.2. Introducing New Reasoning Capabilities 21](#_Toc338688626)

[5.5. The Instantiator View 21](#_Toc338688627)

[5.5.1. Default Variability Implementation Techniques 21](#_Toc338688628)

[5.5.2. Introducing New Variability Implementation Techniques 21](#_Toc338688629)

[5.6. The Attribute Setting Editor 21](#_Toc338688630)

[6. Advanced Software Product Line Engineering with EASy-Producer 22](#_Toc338688631)

[6.1. Staged Configuration and Instantiation 22](#_Toc338688632)

[6.2. Product Line Hierarchies 22](#_Toc338688633)

[6.3. Multi Software Product Lines 22](#_Toc338688634)

[7. Known Bugs 23](#_Toc338688635)

[A. Appendix 24](#_Toc338688636)

[A.1. Running Example IVML-File 24](#_Toc338688637)

# Introduction

EASy-Producer[[1]](#footnote-1) is a Software Product Line Engineering (SPLE) tool which facilities the most recent trends and concepts in SPLE, such as large-scale Multi-Software Product Lines (MSPL), product line hierarchies, and staged configuration and instantiation. The focus of this tool is to support these rather complex concepts in an easy-to-use way. Thus, this tool allows developing a first prototypical Software Product Line (SPL) within minutes. However, EASy-Producer is a research prototype for demonstrating new approaches to SPLE in general and, in particular approaches for simplifying the development of SPLs developed by the Software Systems Engineering group (SSE) at the University of Hildesheim.

This document provides a prototypical[[2]](#footnote-2) user guide that introduces the reader to the concepts and capabilities of EASy-Producer. In Section 2, we will give a brief overview on the SPLE concepts supported by EASy-Producer. This will include introductions to the concepts of SPLE in general, staged configuration and instantiation, MSPL, and product line hierarchies. Further, we will discuss the corresponding approaches in EASy-Producer that were developed for easing the application of these concepts.

Section 3 will give guidance for the first steps with EASy-Producer. This section includes the mandatory prerequisites, the installation guide, and additional recommendations for running the tool successfully.

In Section 4, we will introduce EASy-Producer in terms of describing the development of a first prototypical SPL and the derivation of a product line product. This will cover all aspects of SPL development ranging from creating a new product line project in EASy-Producer, defining a variability model and implementing the corresponding product line artifacts, to the derivation, configuration, and instantiation of a specific product. While the purpose of this section is to describe and illustrate the basic application of EASy-Producer, we will not discuss technical details of the tool at this point. This will be part of the next section.

Section 5 will describe EASy-Producer in detail. This includes detailed descriptions of the individual editors and views as well as discussions on the underlying conceptual decisions and technical capabilities.

In Section 6, we will discuss advanced SPL concepts, like MSPL, staged configuration, etc., and how these concepts can be applied using EASy-Producer.

Finally, Section 7 will summarize known bugs and current workarounds.

# Software Product Line Engineering at a Glance

The EASy-Producer supports basic Product Line Engineering and also staged configuration and Multi Software Product Lines or any combination of these techniques. In the next three sections we will give a short introduction of these concepts.

## Basic Software Product Line Engineering

Software Product Line Engineering (SPLE) is a software development approach which focuses on the extensive reuse of artifacts involved or produced in the software lifecycle. The overall goal of SPLE is to provide a high degree of automation for the configuration and adaptation of product variants. This approach reduces the development effort and costs as well as the time-to-market while increasing the overall software quality.

A Software Product Line (SPL) is a set of related software products which are developed based on a common infrastructure but differ with respect to their provided functionalities. These differences are called variabilities.

## Staged Configuration and Instantiation

Staged configuration and especially staged instantiation are approaches for facilitating partial derivation of product assets. These partial instantiated assets can still contain open variabilities while other variabilities are already bounded. Thus, the configuration can be connected in arranged series. This technique can be used to support different stakeholder/user groups or to create a common basis for related sub-sets of a product line. See Figure 1 for an illustrative example.

Figure 1: Example for staged configuration

## Multi Software Product Lines

Multi Software Product Lines (MSPLs) are able to compose several (independent) product lines to form new products (or product lines). While forming an MSPL, the variability models of the single product lines are combined to an integrated variability model. Derived products can contain instantiated assets from all combined product lines. See Figure 2 for an illustrative example.

Figure 2: Example for a Multi Software Product Line.

# Installation

In this section, we will describe the installation of EASy-Producer. In order to guarantee a successful installation, we will introduce a set of mandatory prerequisites. This will be part of Section 3.1 and sets up the environment in which EASy-Producer is intended to be executed. In Section 3.2, we will describe the installation of the tool in a step-wise manner using the Eclipse update site mechanism and the EASy-Producer update site. Finally, Section 3.3 will give some useful recommendations.

## Prerequisites

EASy-Producer is developed as an **Eclipse[[3]](#footnote-3)** plug-in and requires **Xtext[[4]](#footnote-4) version 2.3.1**. Thus, in general, any Eclipse installation with Xtext version 2.3.1 is fine for installing and running EASy-Producer. However, we cannot guarantee that any combination of Eclipse and Xtext version 2.3.1 will work with EASy-Producer. Thus, we propose the following Eclipse versions to be tested with EASy-Producer (and Xtext version 2.3.1):

* Eclipse 3.6 (Helios)
* **Eclipse 3.7 (Indigo)**
* Eclipse 4.0 (Juno)

We recommend using Eclipse 3.7 (Indigo) as this is the most exhaustively tested version of Eclipse with EASy-Producer. Download an Eclipse package from <http://www.eclipse.org/downloads/>.

Further, Xtext version 2.3.1 has to be installed in the newly downloaded Eclipse instance. Typically, this is installed automatically when installing EASy-Producer due to plug-in dependencies. However, we encountered situations in which these dependencies were not automatically resolved. In such a case, install Xtext manually as follows:

* **Define a new update site** in Eclipse using the following location (if you are not familiar with defining new Eclipse update sites, please consider Section 3.2): <http://download.eclipse.org/modeling/tmf/xtext/updates/composite/releases/>
* **Select the additional plug-ins** to be installed. This will be “*Xtext-2.3.1*” only. All other dependencies will be resolved automatically. **IMPORTANT**: If the dependencies will not be resolved automatically or you encounter problems regarding missing “*EMF*”-plug-ins, please download and use **Eclipse Modeling Tools** instead of any other Eclipse packages.

## Installation: Step by Step

The SSE group hosts an EASy-Producer update site for easy installation and updates. Thus, the first step for installing EASy-Producer is to define a new update site in Eclipse. For this purpose, start Eclipse and open the *Install New Software* dialog by clicking *Help 🡪 Install New Software…* as shown in Figure 3:



Figure 3: Open the “Install New Software” dialog

The *Install* Dialog will appear (cf. Figure 4). In this dialog, a new location for available software has to be added. Thus, click on the *Add…* button in the upper right location of the dialog.



Figure 4: Add a new location for software updates

The *Add Repository* dialog requires the definition of a name for the new update site and a location as illustrated in Figure 4. The name is up to the user. For example, enter “*EASy-Producer update site*”. The location is the URL of the update site:

EASy-Producer update site: <http://projects.sse.uni-hildesheim.de/easy/>

Finish the definition of the new update site by clicking the *OK* button of the *Add Repository* dialog.

The *Install* Dialog will now contain multiple categories. If you are installing EASy-Producer for the first time and do not know which features to select, expand the *Quick Installation of EASy-Producer* category and select the *EASy-Producer Quick Installation* feature. This will install all required components automatically. For more experienced users, select the categories and features as needed and click the *Next* button. Follow the steps for installing EASy-Producer (accept the license agreement and ignore the security warning for installing software of unsigned content, etc.), and restart Eclipse as prompted.

Finally, you have successfully installed the EASy-Producer.

## Recommendations

In order to avoid memory problems while using EASy-Producer, we recommend increasing the memory of the Eclipse application in which EASy-Producer is executed. The memory problems are due to Xtext which requires more memory than defined in a typical Eclipse configuration.

Open the “*eclipse.ini*” file in your Eclipse directory and enter the following parameters at the end of the file:

-vmargs

-Xms40m

-Xmx512m

-XXMaxPermSize=128m

# Getting Started: Product Line Engineering is EASy

In this section, we will focus on the roles of a domain engineer and an application engineer in order to illustrate the application of EASy-Producer based on a running example. We will prototypically model and implement the variability of a content-sharing platform, which allows the user to upload, annotate, release and share content of various types. Section 4.1 will describe this example in detail. In Section 4.2, we will adopt the role of a domain engineer and describe the definition of a SPL from which multiple variants of the content-sharing platform can be derived. This includes the definition of the variability model using the INDENICA[[5]](#footnote-5) Variability Modelling Language (IVML) and the implementation of these variabilities in source code. In Section 4.3, we will adopt the role of an application engineer and describe the derivation of a specific service platform variant including the variant configuration and the instantiation of the corresponding artefacts.

We will use the following font styles throughout this section to illustrate and distinguish between actions, active tool elements, and added input:

* EASy-Producer (as well as Eclipse) provides multiple editors, wizards, etc. In order to identify the **active tool element** currently in use, it will be highlighted using bold font.
* All *actions* that will be performed will be highlighted using italics font.
* All input to EASy-Producer will be illustrated in Courier New.

Please note that we will not discuss conceptual or technical details of the tool in this section. This will be part of the detailed description of EASy-Producer in Section 5.

## Running Example

In this section, we introduce a running example which we will use throughout Section 4 to illustrate the basic application and capabilities of EASy-Producer. In this example a content-sharing platform will be developed in terms of a SPL. A content-sharing platform allows its users to upload, annotate, release and share content of various types. In this example, concrete applications may differ with respect to:

* The supported content types such as text, video, audio, 3D content, or binary (large) objects (BLOBs).
* The hosting infrastructure which consists of a) a web container being responsible for serving the content and b) the database, which stores user and content data.
* The deployment target which may either be a traditionally hosted server, or a cloud environment. The cloud environment may be private, like a local installation of the Eucalyptus[[6]](#footnote-6) cloud software or public, in this example we will allow connections to Amazon[[7]](#footnote-7) or Azure[[8]](#footnote-8) cloud.

Without going into functional details of the content-sharing platform, the variabilities introduced by content types, web container, database and deployment target allow to derive a large number of different platform instances. However, some dependencies exist that restrict the selection of variants to be part of a specific platform instance. These restrictions and dependencies will be modelled in terms of constraints in the variability model in Section 4.2.1:

1. At least one content type must be present as otherwise the content-sharing platform is useless.
2. To ensure acceptable quality of service, the maximum bit rate for video content on the Tomcat web container is 128 kBit/s.
3. The combination of supported content types may be restricted based on the capabilities of the web container or the deployment platform, e.g. due to load problems only a limited number of content types may be available on the traditional deployment target.
4. Some content types may be served by a separate web container in order to configure a simple load balancing mechanism, for example 3D content should be served by a JBoss server. As a further extension, a web container may be configured to retrieve its content from a specific database.
5. Content types may be transformed and the result may be shared. Such transformations should be configured in terms of configuration chains, such as the textual representation of the audio track of a video. As transformations may be resource-consuming and, thus, affect the performance, on the traditional platform only simple and resource saving implementations should be deployed while resource-consuming high-quality transformations may be used on the cloud platforms.

This content-sharing platform product line will be developed in the following sections using EASy-Producer. In particular, we will focus on the variability modelling, the variability implementation and the derivation of a specific platform instance.

## Defining a New Base Service Platform

In this section, we will describe the process of defining the variability of a (base) service platform (a SPL) using EASy-Producer from the perspective of a domain engineer. We will start with the creation of a new product line project in EASy-Producer, define the configuration space in terms of an IVML variability model, and implement the variabilities using a variability implementation technique. The resulting base service platform (the product line project) will be the basis for the derivation of different content-sharing platforms by an application engineer.

The first step towards a product line definition in EASy-Producer is to define a new product line project. For this purpose, start the Eclipse application with the already installed EASy-Producer tool (see Section 3 for installation details). Start the **New Project Wizard** by opening *File 🡪 New 🡪 Project*. Expand the EASy-Producer category and select the entry **New EASy-Producer Project**. This opens the **Product Line Project Wizard** that requires the definition of a name for the new product line project. In our example, we will use PL\_Content\_Sharing as the name of our prototypical product line. EASy suggests to name the newly created project with a prefix (PL\_) However, it is not necessary to keep this prefix. Enter the name and click the *Finish* button. The product line project will be created and EASy-Producer will automatically open the **Product Line Editor** as illustrated in Figure 5.

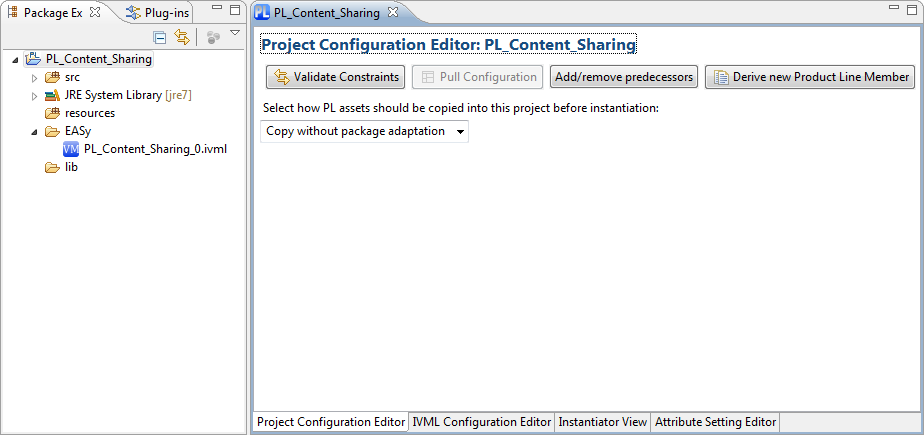


Figure 5: Running Example – The Product Line Editor.

The **Product Line Editor** is the central editor in EASy-Producer as it provides the basic information about a SPL (or a product) as well as the capabilities to derive, configure, and instantiate a product using the different tabs shown in Figure 5. For this purpose, the configuration space (variability model) and the implementation space (variability implementation) must be defined. We will describe both definitions in detail in the next two subsections.

### Configuration Space Definition

A variability model defines the valid configuration space of a specific SPL. The variabilities are implemented in the artefacts. In EASy-Producer, we use IVML for defining a variability model and, thus, the configuration space of the content-sharing platform. This model will be the basis for configuring individual service platforms in terms of defining valid value combinations for the configuration space elements (the IVML decision variables).

In EASy-Producer, each product line project comes with its own IVML-file, which can be opened and edited using the **IVML-Editor**. The IVML-file is located in the **EASy**-**folder** of the project. The name of the file is composed of the name of the product line and the version number (here initially “0”). In our example, double-click the file **PL\_Content\_Sharing\_0** in order to open the **IVML-Editor**.

By default, each IVML-file has a mandatory project element and a mandatory version number as shown in Figure 6. The project element is the top-level element of each IVML file and identifies the configuration space of a certain software project (product line or product). The version element defines the current state of evolution of a project and, thus, identifies a specific (state of a) project. The default version is “v0”.

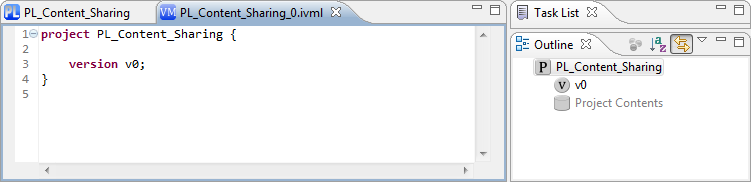


Figure 6: Running Example – The IVML Editor.

We characterize the configuration space of the variant-enabled content-sharing base platform by specifying the variability model in IVML. Figure 7 shows a snippet of the variability model (the complete model can be found in the appendix A.1). First, we define several enumerations that represent the different content types, container types, etc., which an application may support in general (lines 5-8). These enumerations are the basis for specifying the type, for example, of a specific content (lines 10-12). The basic content compound must be refined in order to represent the specific configuration options for Video, 3D (ThreeD), and BLOB contents (lines 14-27). The other compounds are modelled according to the running example (cf. Section 4.1). As indicated in the outline on the right side of Figure 7, the two types Application and TargetPlatform include decision variables of the previously defined (compound) types representing the complete set of configuration options for the content-sharing base platform. Thus, two variables (one of type Application and one of type TargetPlatform) are defined as the main decision variables for configuring a specific content-sharing platform variant. These variables will also be displayed in the **IVML Configuration Editor** tab of the **Product Line Editor**. We will discuss this editor in detail in the process of product configuration in Section 4.3.1.

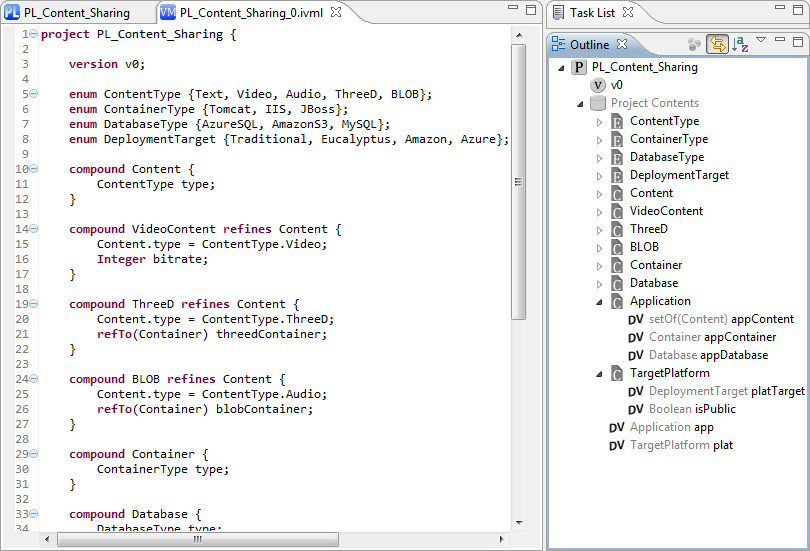


Figure 7: Running Example – The Variability Model (snippet).

Finally, the variability model, and, thus, the configuration space of the content-sharing application is defined. We will use this model in Section 4.3.1 for configuring a specific content-sharing platform variant. However, in the next section we will first discuss the implementation of the variabilities. This includes the relation of the decision variables to the implementation in order to automatically instantiate different platform variants.

### Implementation Space Definition

The implementation space of a specific SPL represents all variable artefacts that can be instantiated according to a specific configuration. The actual implementation of these artefacts depends on the applied variability implementation techniques (VITs). A VIT is a specific approach to realize variability, e.g., using pre-processor directives, aspects, or any other techniques. In EASy-Producer different VITs can be applied and combined. However, each VIT is realized by an individual instantiator, which actually applies the VIT (we will discuss the instantiators in detail in Section 5.5). In the running example, we will use the Velocity instantiator as it is one of the default instantiators of the basic EASy-Producer installation.

All product line (and product) source code is located in the **src** folder of the product line project as shown in Figure 8. The Velocity instantiator provides pre-processor functionality to Java and can be applied in terms of adding Velocity-specific statements to plain Java code. In lines 5 and 6 of Figure 8, the deployment platform and the public switch will be defined accordingly to the values of platTarget and the isPublic variables (cf. the variability model in Figure 7). Both variables are nested variables of the platform variable plat. Thus, they are accessed using “-”-notation. In order to guarantee that Velocity will find these variables, the instantiator requires a dollar-sign in front of the variable declarations in the code (for more details about Velocity see Section 5.5.1).

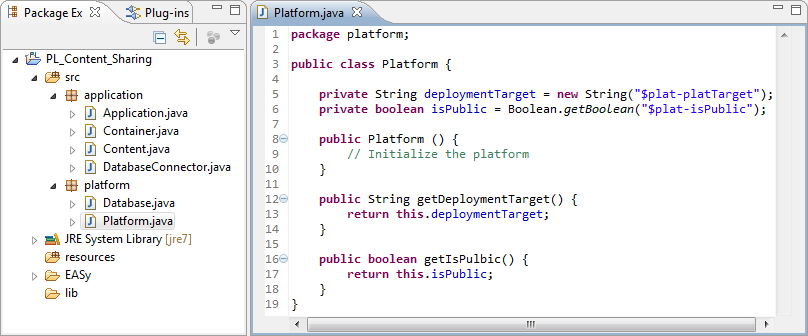


Figure 8: Running Example – The Variability Implementation (snippet).

The next step is to define the instantiator for instantiating the artefacts. Open the **Product Line Editor** by *right clicking on the product line project* and select *Edit Productline* in the context menu. Switch to the **Instantiator View** tab of the **Product Line Editor,** which is illustrated in Figure 9. In our example, we will select **VelocityTransformer** as the instantiator for the entire product line by *selecting the corresponding entry* in the drop-down menu in the upper part of the **Instantiator View**. Clicking the *Add Instantiator* button will add the Velocity instantiator to the product line project. Select the new (instantiator) entry in the list on the left side of the view. This will display the currently selected artefacts that the Velocity instantiator will instantiate. Of course, at this point we have to select the artefacts by checking the *checkboxes* for all files in the **src** folder. Please note that it is also possible to define multiple instantiators, where each one may be in charge of a subset of the product line artefacts, or define multiple instantiators for the same files. The latter case is required if a single artefact is implemented using multiple, different variability implementation techniques, i.e. to enable some variable parts to be resolved at compile time first (for example, using pre-processer directives), while other part can be resolved later at runtime (for example, using aspects). In such a case, the *Calculate files that will be instantiated multiple times* button can be used to identify files that will be instantiated by more than one instantiator.

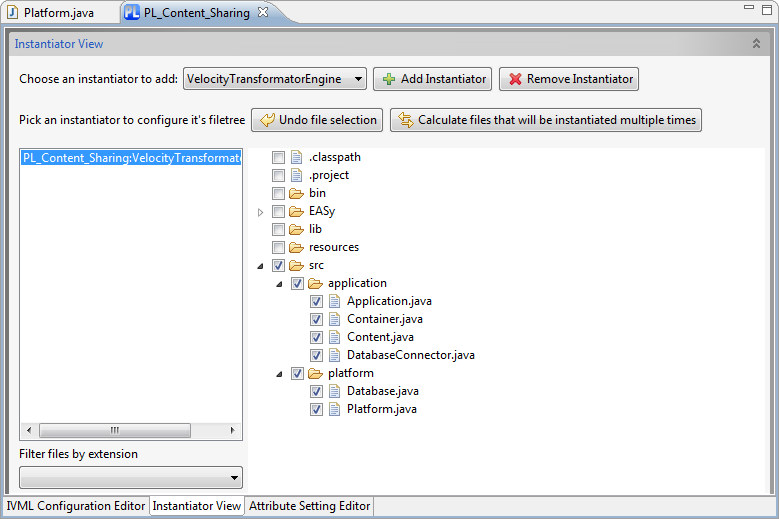


Figure 9: Running Example – The Instantiator View.

Finally, the implementation space is defined and an instantiator is assigned to instantiate content-sharing application variants accordingly to a configuration. On this basis, we will derive a new product from this product line in the next section.

## Deriving a Domain-Specific Service Platform

In this section, we will describe the process of deriving a new domain-specific platform from a software product line defined in EASy-Producer. We will adopt the perspective of an application engineer and start with the derivation of a new product line member[[9]](#footnote-9) (in this case, the product project), configure the product based on the variability model defined in Section 4.2.1, and instantiate the product line artefacts accordingly. This will result in a specific content-sharing application variant with the desired functionalities ready for use.

The first step towards an instantiated domain-specific platform is to derive a new member from the previously defined base platform product line. For this purpose open the **Product Line Editor** by right clicking on the *product line project* and select *Edit Productline* in the context menu. In the **ProjectConfiguration Editor** tab click the *Derive new Product Line Member* button, define a name for the new member, and click the *Ok* button. In our running example, we will use Audio\_Sharing\_App as the name of the new member. A new product line project will be created and the corresponding **Product Line Editor** will open automatically as shown in Figure 10.

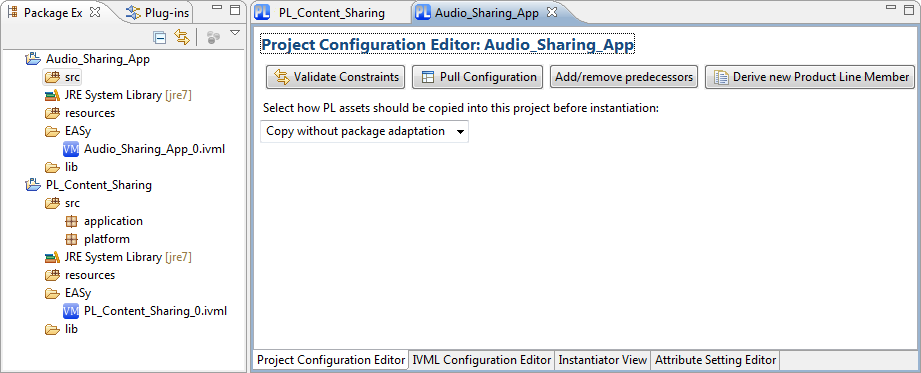


Figure 10: Running Example – The Product Derivation.

In the new product line member project, we will configure the desired functionalities of our specific audio content-sharing platform. This configuration will be used to finally instantiate the domain-specific platform. We will describe both steps in detail in the next two sections.

### Configuration of a Domain-Specific Service Platform

A product configuration (in this example the configuration of the domain-specific service platform) is a set of configured elements. In IVML configured elements are specified by assigning specific values to the elements in the configuration space, i.e. the decision variables, the attributes, etc. The validity of a configuration is checked against the constraints of the variability model using the built-in reasoning mechanism. The valid product configuration provides the basis for the (automated) instantiation of the corresponding product artefacts.

EASy-Producer provides two ways of configuring the elements of an IVML variability model: either use the **IVML Editor** by double-clicking the *IVML file* of the derived product line member (in our example the Audio\_Sharing\_App\_0.ivml file) in order to configure the elements of the imported project (the product line project) manually, or use the **IVML Configuration Editor** tab of the **Product Line Editor**. In our example, we will use the **IVML Configuration Editor**. This eases the configuration task as it includes all configurable elements of the imported project and provides the possible values for each of these elements automatically (we will discuss the configuration editor in detail in Section 5.4). Figure 11 illustrates the **IVML Configuration Editor,** including the configurable elements of our audio content-sharing application.

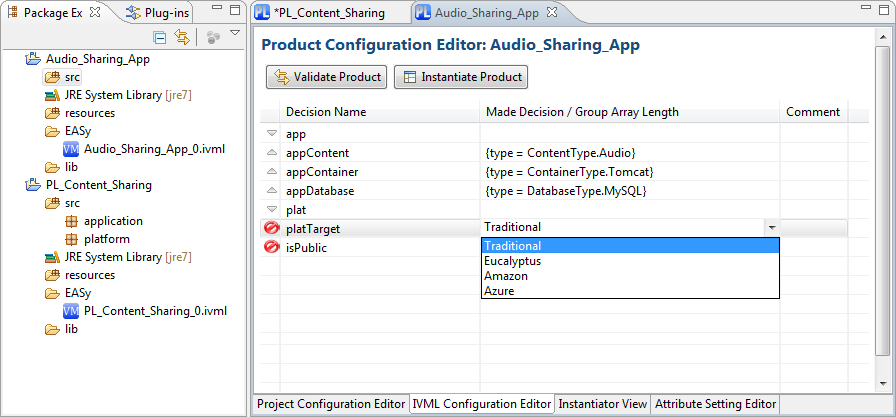


Figure 11: Running Example – The IVML Configuration Editor.

The next step is to check whether the configuration is valid. For this purpose, click on the *Validate Product* button of the **IVML Configuration Editor**. This executes the built-in IVML reasoning. If the product is valid, it is ready for instantiation. If it is not valid, the configuration must be revised in order to guarantee that the resulting product will work appropriately. In case of an invalid configuration, EASy-Producer will issue a description of the configuration problem and propose a possible error location in the current configuration. Please note that this kind of user support is still under development and may not work appropriately in every situation in the current release of EASy-Producer. However, we are working on this kind of support as part of future releases to ensure scalability to large and complex configuration problems.

Finally, the product is configured and ready for instantiation.

### Instantiation of a Domain-Specific Service Platform

Product instantiation describes the process of resolving the variability of product line artefacts according to a product configuration. This process results in the product artefacts that are mostly variation-free and ready to use. However, in some situations it is desired to resolve some of the variabilities at a later point in time, for example, at initialization time or runtime. In such a case, the instantiation process will leave these variabilities as-is.

EASy-Producer provides a fully automated instantiation process, which is based on the variability model, the current configuration and the selected instantiators. We defined this information in the previous sections, such as the implementation space and instantiator definition (cf. Section 4.2.2) and the product configuration (cf. Section 4.3.1). This relies in turn on the configuration space definition (cf. Section 4.2.1). Thus, the last step is to define how the generic artefacts defined for the variant-enabled base platform will be turned into artefacts of the instantiated domain-specific platform. Currently, EASy-Producer offers two ways: 1) to instantiate the artefacts in terms of their structure, i.e., by taking over existing namespaces, packages and class / file names; 2) to add additional namespaces for distinguishing the artefacts combined from different product lines. The creation of namespaces as part of deriving instantiated product line artefacts is related to the concept of Multi-Software Product Lines (MSPL). We will discuss this and other advanced SPLE concepts in detail in Section 6.

However, in our example, we will use the first (simple) mechanism relying on the current artefact structure. Thus, switch to the **Project Configuration Editor** tab and use the drop-down menu to select the *Copy without package adaptation* entry as shown in Figure 12.

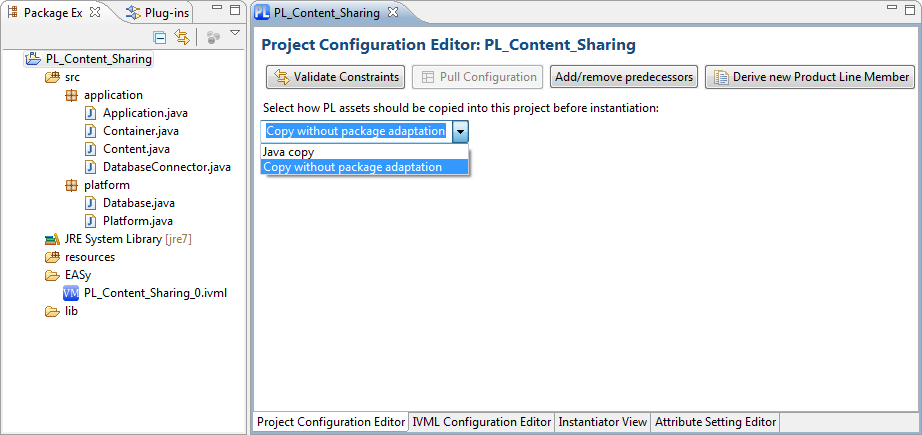


Figure 12: Running Exmaple – The Copy-Mechanism.

Finally, the only remaining activity is to click the *Instantiate Product* button in the **IVML Configuration Editor**. This will yield the instantiated artefacts from the product line project and inserts them into the product project by resolving the variabilities.

# EASy-Producer in Detail

**Under Development**

**Under Development**

**Under Development**

**Under Development**

**Under Development**

**Under Development**

## The Product Line Project Structure

## The Product Line Editor

## The Project Configuration Editor

## The IVML Configuration Editor

### Default Reasoning Capabilities

### Introducing New Reasoning Capabilities

## The Instantiator View

### Default Variability Implementation Techniques

### Introducing New Variability Implementation Techniques

## The Attribute Setting Editor

# Advanced Software Product Line Engineering with EASy-Producer

**Under Development**

**Under Development**

**Under Development**

**Under Development**

**Under Development**

**Under Development**

## Staged Configuration and Instantiation

## Product Line Hierarchies

## Multi Software Product Lines

# Known Bugs

The following bugs and problems are known. We are currently working on solutions.

* The variability model defined in the IVML-editor disappears. Workaround: Do not run both editors (the product line-editor and the IVML-editor) for the same project simultaneously. This may cause overriding your previously defined model or the configuration in one of the editors by the other one. Only use one editor at the same time (i.e. close the IVML-editor before opening the product line-editor).
* The IVML Configuration Editor of the EASy product line-editor does not display the decision variable defined in the IVML-editor. Workaround: After defining the variability model, close the IVML-editor, close the current project and open it again. The IVML Configuration Editor will now display the decision variables. Please note that deriving a product from a product line project in which the variables will not be displayed in the IVML Configuration Editor will yield a corrupted product. This corrupted product must be deleted and derived again (after checking whether the decision variables are available in IVML Configuration Editor of the product line project).
* The “Instantiate”-button is disabled in the product-project. Workaround: Check whether an instantiator is defined in the parent product line-project. Typically this error points at a not defined instantiator. In such a case, the derived product must be deleted and derived again.
* In some cases, IVML elements of an imported project are marked as unknown (errors). Workaround: Type something in the IVML-editor and save the file.
* In some situations, you may encounter performance problems or, in the worst case, freezing eclipse. If you configured the eclipse.ini file as described in Section 3.3, restart eclipse.

In case that you encounter any further bugs feel free to contact us in terms of writing a mail to your current contact person of the SSE group.

Thank you!

1. Appendix
2. Running Example IVML-File

**project** PL\_Content\_Sharing {

**version** v0;

**enum** ContentType {Text, Video, Audio, ThreeD, BLOB};

**enum** ContainerType {Tomcat, IIS, JBoss};

**enum** DatabaseType {AzureSQL, AmazonS3, MySQL};

**enum** DeploymentTarget {Traditional, Eucalyptus, Amazon, Azure};

**compound** Content {

ContentType type = ContentType.Audio;

}

**compound** VideoContent **refines** Content {

**Integer** bitrate;

Content.type = ContentType.Video;

}

**compound** Container {

ContainerType type = ContainerType.Tomcat;

}

**compound** Database {

DatabaseType type = DatabaseType.MySQL;

}

**compound** Application {

Content appContent;

Container appContainer;

Database appDatabase;

}

**compound** TargetPlatform {

DeploymentTarget platTarget;

**Boolean** isPublic;

}

**compound** ThreeD **refines** Content {

**refTo**(Container) threedContainer;

Content.type = ContentType.ThreeD;

}

**compound** BLOB **refines** Content {

**refTo**(Container) blobContainer;

Content.type = ContentType.Audio;

}

Application app;

TargetPlatform plat;

plat.platTarget = DeploymentTarget.Traditional **implies**

app.appDatabase.type = DatabaseType.MySQL;

plat.platTarget = DeploymentTarget.Eucalyptus **implies**

app.appDatabase.type = DatabaseType.AmazonS3;

plat.platTarget = DeploymentTarget.Amazon **implies**

app.appDatabase.type = DatabaseType.AmazonS3;

plat.platTarget = DeploymentTarget.Azure **implies**

app.appDatabase.type = DatabaseType.AzureSQL;

}

1. EASy is an abbreviation for Engineering Adaptive Systems. [↑](#footnote-ref-1)
2. Please note that this document is currently under development and will evolve with the evolution of the tool. Changes are documented in the version section. [↑](#footnote-ref-2)
3. Eclipse website: [www.eclipse.org/](http://www.eclipse.org/) [↑](#footnote-ref-3)
4. Xtext website: <http://www.eclipse.org/Xtext/> [↑](#footnote-ref-4)
5. INDENICA is an EU-funded project in which the variability modeling language of EASy-Producer was initially designed and developed. However, this language is not INDENICA-specific but was designed with further requirements from research and industry in mind. For more information regarding INDENICA please visit the INDENICA website: <http://indenica.eu/> [↑](#footnote-ref-5)
6. Eucalyptus website: <http://open.eucalyptus.com/> [↑](#footnote-ref-6)
7. Amazon cloud website: <http://aws.amazon.com/de/ec2/> [↑](#footnote-ref-7)
8. Azure website: <http://www.microsoft.com/windowsazure/> [↑](#footnote-ref-8)
9. In EASy-Producer, we do not distinguish between a product line infrastructure and a final product. Both are simply projects that may contain more or less variability (in case of a product none). [↑](#footnote-ref-9)