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

Development Version 1.5.2 of 03.06.2021

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, and appendix initially added.
0.4	30.12.2012	Section 5 updated (screenshots and descriptions added).
0.5	04.03.2013	Section 3 updated (inclusion of Xtext features in EASy update site).
0.6	02.09.2013	Section 4 updated (inclusion of VIL)
1.0	27.09.2013	Version 1.0 completed (reference to other developers guide, IVML and VIL language specification, corrected spelling, updated figures)
1.1	11.07.2014	FAQ (Section 6) added.
1.2	13.02.2015	FAQ sections for Maven and the SSE reasoner added
1.3	14.07.2015	Migration to Xtext version 2.5.3, Examples, Updated installation of EASy-Producer
1.4	20.07.2015	Described example: EASyDemoCommands
1.5	29.07.2015	Updated Examples section.
1.5.1	08.01.2018	Term / line number updates (reported by M. Keunecke)
1.52	03.06.2021	Extended .text files

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 Apache License, Version 2.0.

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

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

Table of Contents

1. Introduction

EASy-Producer[1] is a Software Product Line Engineering (SPLE) tool which facilitates 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. Further, 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 live-document provides a 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.

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

In Section 4, we 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 artefacts, 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 all 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 of the tool. In Section 6, we explain how to run some shipped examples to learn the basic concepts of EASy-Producer. Finally, Section 0 provides a FAQ (Frequently Asked Questions).

[1] EASy is an abbreviation for Engineering Adaptive Systems.

2. Software Product Line Engineering at a Glance

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 to these concepts.

2.1. Basic Software Product Line Engineering

Software Product Line Engineering (SPLE) is a software development approach which focuses on the extensive reuse of artefacts 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.

2.2. Staged Configuration and Instantiation

Staged configuration and especially staged instantiation are approaches for facilitating partial derivation of product artefacts. These partial instantiated artefacts can still contain open variabilities while other variabilities are already bound. 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.

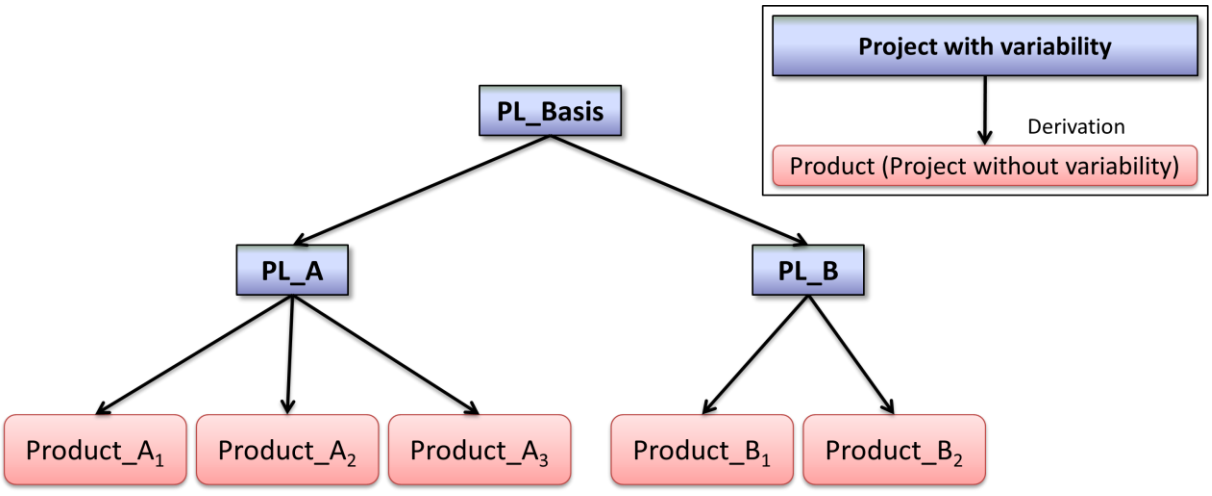


Figure 1: Example for staged configuration

2.3. 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 artefacts from all combined product lines. See Figure 2 for an illustrative example.

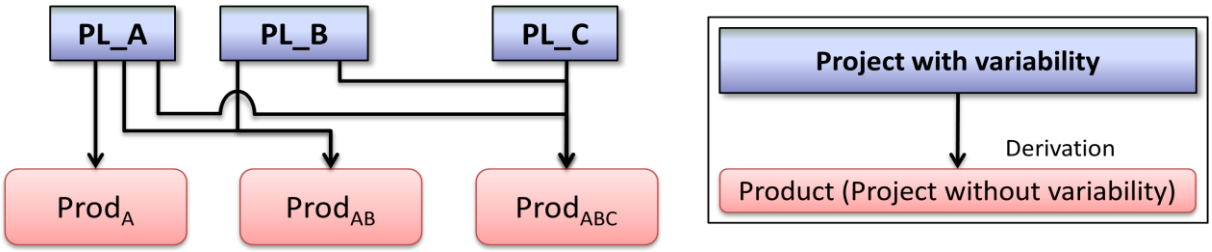


Figure 2: Example for a Multi Software Product Line.

3. Installation

In this section, we 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 in which we set up the environment in for EASy-Producer. In Section 3.2, we describe the installation of the tool in a step-wise manner using the Eclipse update site mechanism and the EASy-Producer update site. EASy-Producer can alternatively used as a command line tool, which does not require an Eclipse installation. This facilitates the usage of EASy-Producer as part of a Continuous Integration setup. In Section 3.3, we show how to install EASy-Producer as a command line tool, outside of Eclipse. Section 3.3 introduces additional guides and specifications for EASy-Producer.

3.1. Prerequisites

EASy-Producer is developed as an **Eclipse**[2] plug-in and requires **Xtext**[3] **version 2.5.3**. Thus, in general, any Eclipse installation with Xtext version 2.5.3 is fine for installing and running EASy-Producer. However, we cannot guarantee that any combination of Eclipse and Xtext version 2.5.3 will work with EASy-Producer. Thus, we propose the following Eclipse versions as they are tested with EASy-Producer (and Xtext version 2.5.3):

- Eclipse 4.3 (Kepler)
- Eclipse 4.4 (Luna)
- Eclipse 4.5 (Mars)

We recommend using **Eclipse 4.3 (Kepler)** as this is the most exhaustively tested version of Eclipse with EASy-Producer. Download an Eclipse (e.g. Eclipse IDE for Eclipse Committers) package from http://www.eclipse.org/downloads/.

Further, Xtext version 2.5.3 has to be installed in the newly downloaded Eclipse instance. It can be installed from EASy-Producer update site when installing EASy-Producer due to specific version of Xtext supported by EASy-Producer.

3.2. 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:

[2] Eclipse website: www.eclipse.org/
[3] Xtext website: http://www.eclipse.org/Xtext/

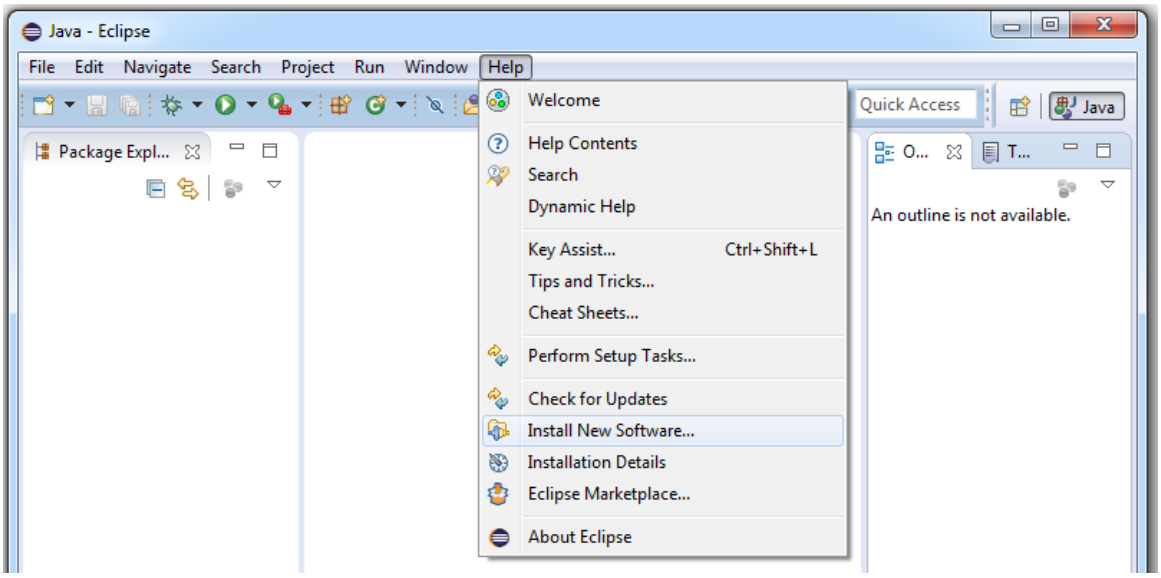


Figure 3: Open the "Install New Software" dialog

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.

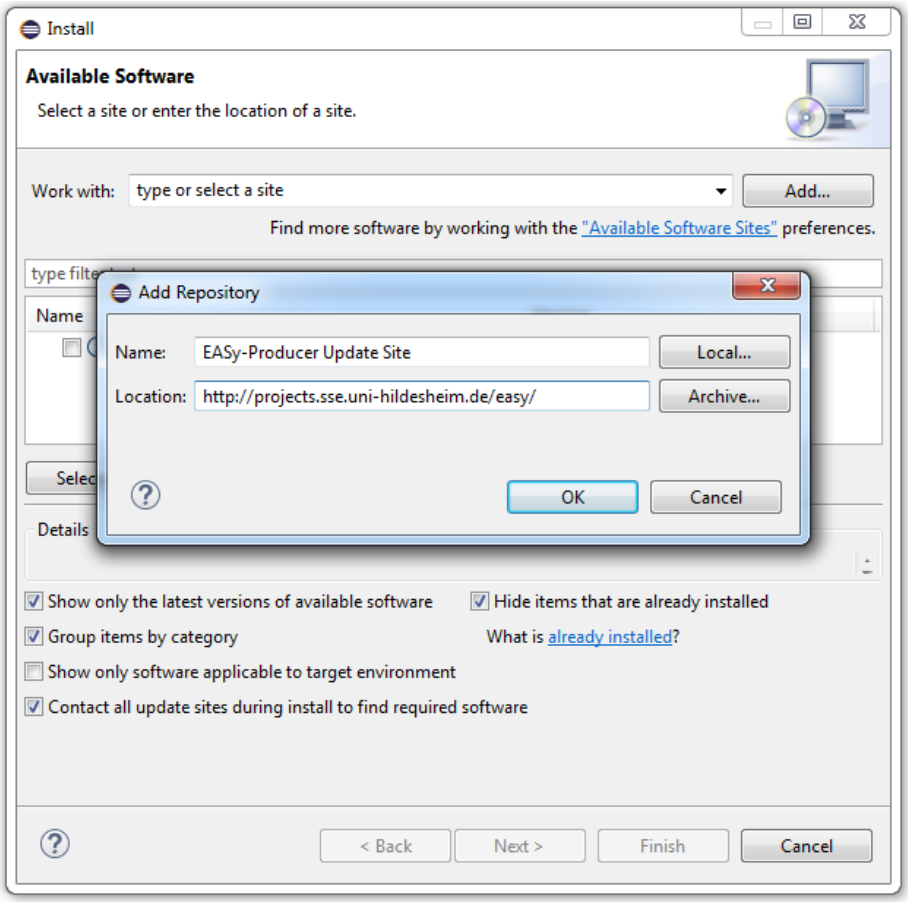


Figure 4: Add a new location for software updates

The *Add Repository* dialog requires for the location of the new update site (and optionally a name) as illustrated in Figure 4. The location is the URL of the 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 select "*EASy-Producer*" inside the *EASy Producer* category and "Xtext Runtime" inside the Xtext-2.5.3 category (cf. Figure 5). Selecting "EASy-Producer Examples" is optional in case you would like to have standard EASy-Producer examples in your Eclipse. This will install all required components (all instantiators) automatically. In Section 6, we briefly describe how to use examples.

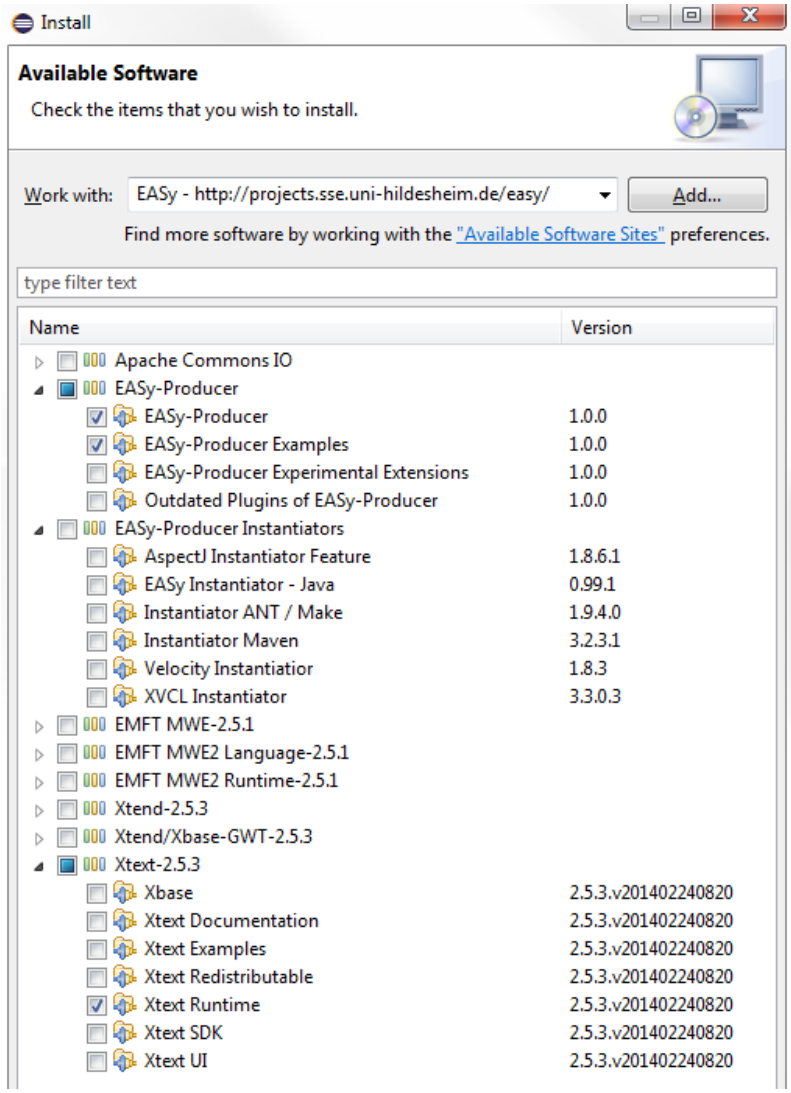


Figure 5: Recommended plug-ins for EASy Producer.

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.

3.3. Installation of the Command Line Tool

EASy-Producer can alternatively be used outside of Eclipse as a standalone command line tool. This facilitates the usage of EASy-Producer as part of continuous integration setting.

The command line tool can be downloaded from http://projects.sse.uni-hildesheim.de/easy/ (cf. Figure 6). The downloaded Zip archive must be unpacked to an arbitrary location. For execution, an installed JRE 6 or higher is needed.

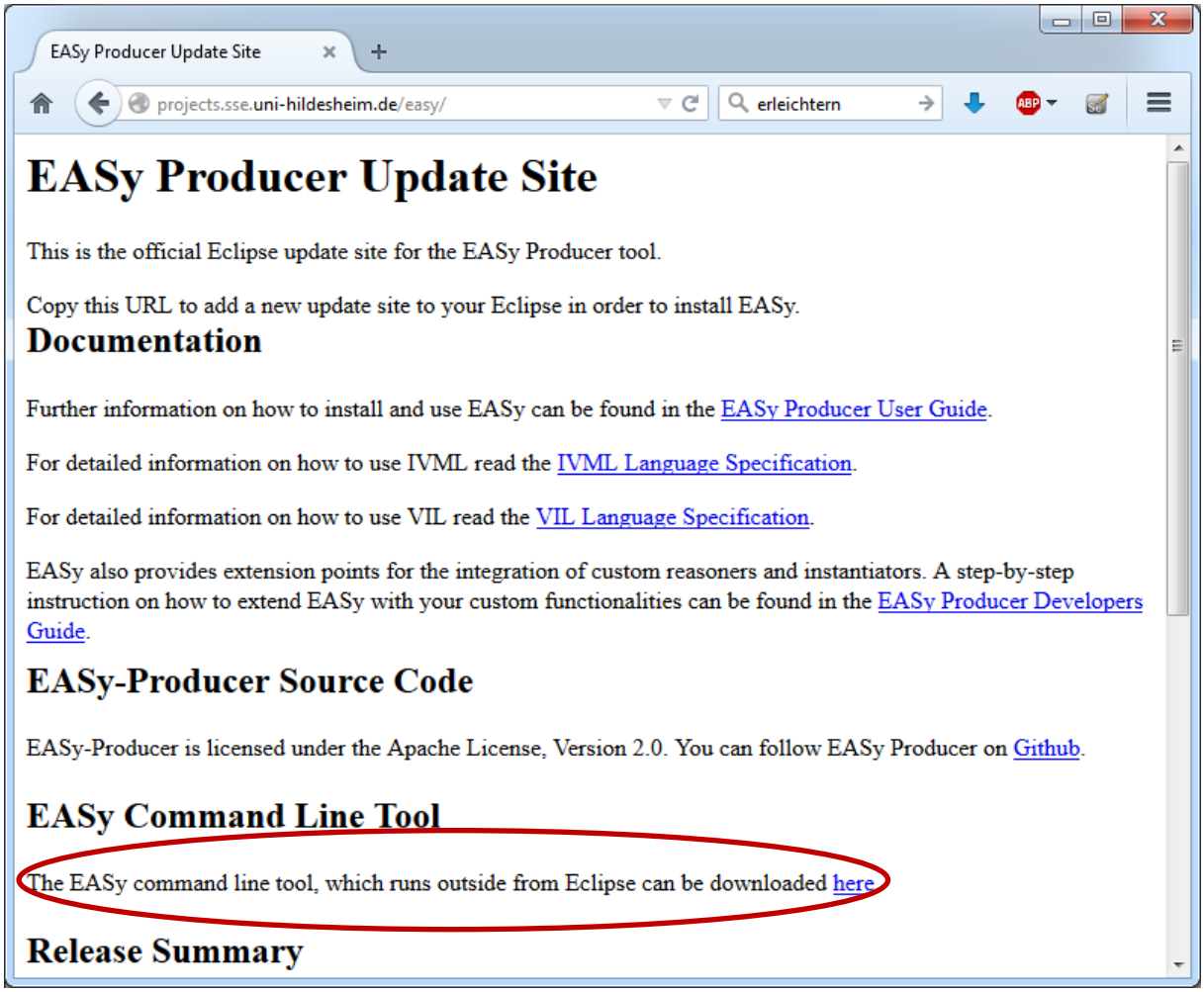


Figure 6: Download link of the EASy-Producer command line tool.

3.4. Further Guides and Specifications

EASy-Producer provides two expressive languages that support the creation of required software product line artefacts:

The **INDENICA Variability Modelling Language (IVML)** is and expressive, textual variability modelling language, which provides basic and advanced modelling capabilities for the definition of variability models. In order to define such a model based on IVML, we provide the IVML language specification. This specification is part of the EASy-Producer installation and can be found in the **Eclipse Help**.

The **Variability Implementation Language (VIL)** is a textual language for the flexible specification of the instantiation process of a software product line. This language consists (beside other parts) of the VIL build language and the VIL template language (VTL). The former language provides modelling elements for the specification of the individual build tasks of the instantiation process, while the latter language supports the definition of templates that can be applied to specific artefacts, for example, to manipulate their content, as part of the instantiation process. The corresponding VIL language specification is also part of the EASy-Producer installation and can be found in the **Eclipse Help**.

Further, EASy-Producer supports the extension of the tool by custom instantiatiors and reasoners. The **EASy-Producer Developers Guide** introduces the reader to the possible extensions and provides a step-wise description of how to extend the tool. This guide can be found in the **Eclipse Help** as well.

The EASy-Producer user guide, the EASy-Producer developers guide, as well as the IVML and the VIL language specification are also available as PDFs on the EASy-Producer update site: http://projects.sse.uni-hildesheim.de/easy/

4. Getting Started: Product Line Engineering is EASy

In this section, we will adopt 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[4] Variability Modelling Language (IVML), the implementation of these variabilities in source code, and the definition of a build script for the instantiation of the generic artefacts using the Variability Implementation Language (VIL). 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 the tool in all details in this section. This will be part of the detailed description of EASy-Producer in Section 5. Further, we will not discuss the IVML and VIL language here. A detailed description of these languages can be found in the corresponding language guides (cf. Section 3.3).

4.1. 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

[4] 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/

Eucalyptus[5] cloud software or public, in this example we will allow connections to Amazon[6] or Azure[7] 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.

4.2. 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. Further, we will define a corresponding build script in VIL to specify the instantiation process of the variable artefacts. 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*

[5] Eucalyptus website: http://open.eucalyptus.com/
[6] Amazon cloud website: http://aws.amazon.com/de/ec2/
[7] Azure website: http://www.microsoft.com/windowsazure/

→ *Project*. Expand the EASy-Producer category and select the entry **EASy-Producer Product Line 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 naming 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 7.

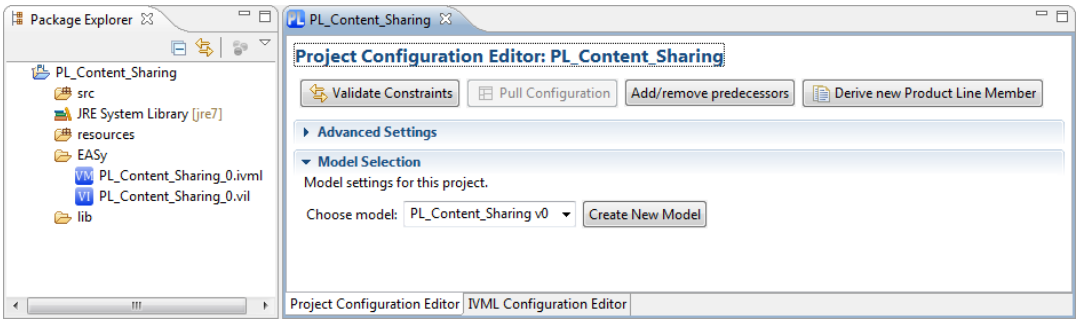


Figure 7: Running Example – The Product Line Editor.

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 7. 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 sections.

4.2.1. 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[8] 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 8. 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".

[8] See the IVML language specification (cf. 3.3) for a detailed description of this language.

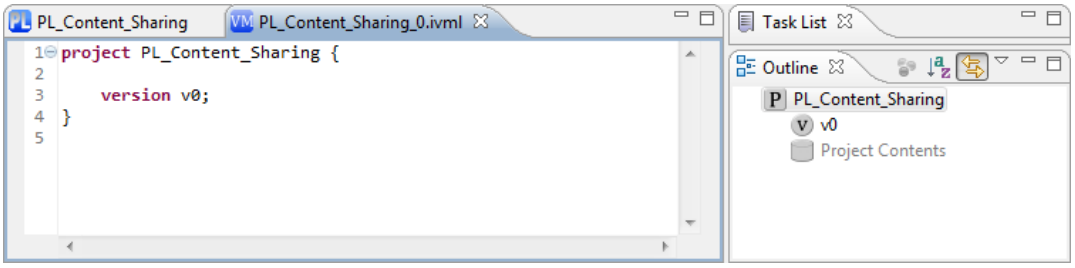


Figure 8: Running Example – The IVML Editor.

We characterize the configuration space of the variant-enabled content-sharing base platform by specifying the variability model in IVML. Figure 9 shows a snippet of the variability model. First, we define several enumerations that represent the different content types, container types, etc., which an application may support in general (lines 4-7). These enumerations are the basis for specifying the type, for example, of a specific content (lines 18-10). The basic content compound must be refined in order to represent the specific configuration options for `Video`, `3D` (`ThreeD`), and `BLOB` contents (lines 12-25). 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 9, 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.

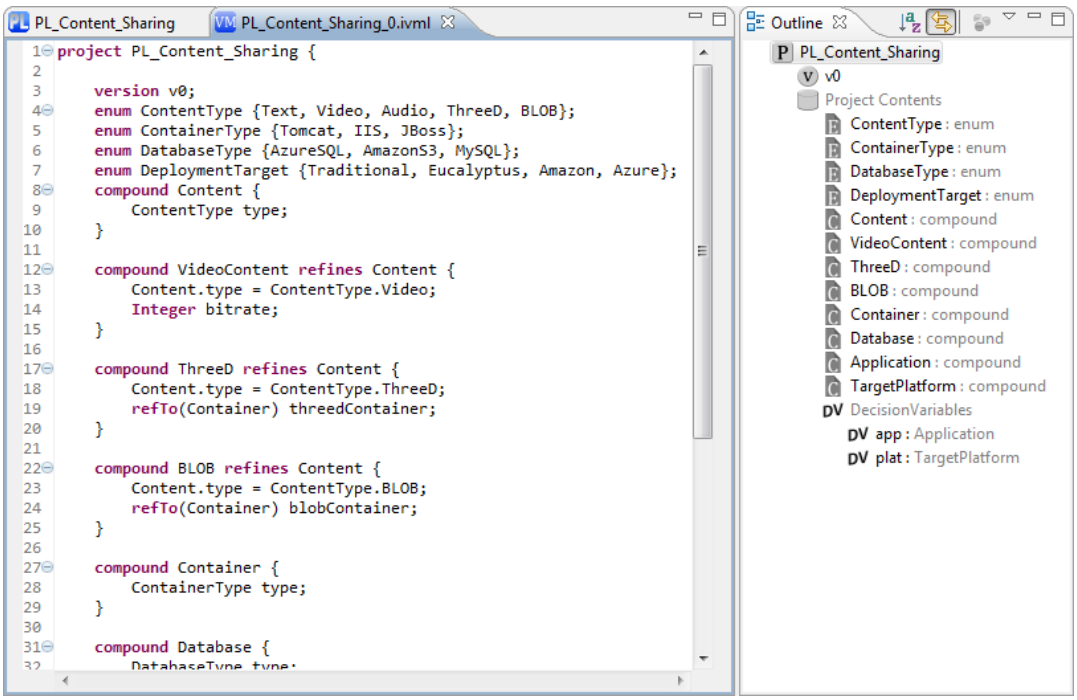


Figure 9: Running Example – The Variability Model (snippet).

In addition to the variability model, we will also define comments for the decision variables in order to support the application engineer in the configuration of a valid product. These

comments can be used as hints for users to fill the model, i.e., they may show up in the UI or they are taken into account by the reasoner to make the reasoning messages more context specific.

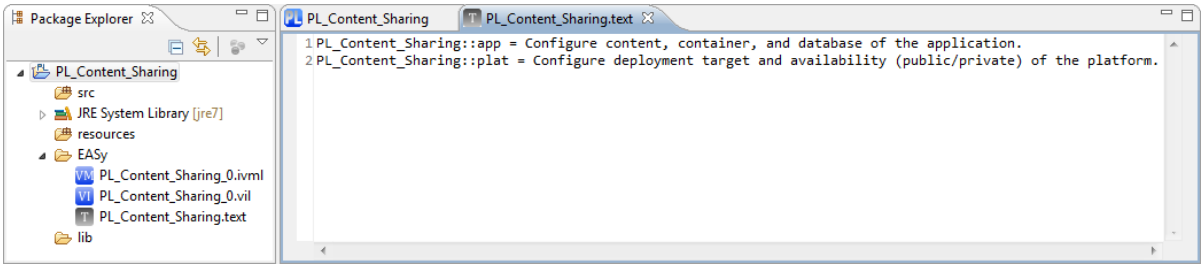
The first step is to create a new text-file in the **EASy-folder** of the product line project. For this purpose, right-click on the **EASy-folder** and select *New → File*. The name of the file must match the name of the variability model (the project-name in IVML) followed by the version number to unambiguously link the comments to the desired decision variables. Thus, enter `PL_Content_Sharing_0.text` as the name[9] of the new file and click the *Finish* button. We will open the new text-file with a simple text-editor to enter the comments. Currently two forms of comment key-value mappings are supported

- Comments for decision variables based on the qualified name of the individual variables. The key for decision variables is the qualified name of the variable, i.e., it starts with the name of the variability model (the project name) followed by "::", the name of the decision variable for which we want to define the comment, and an equal-sign ("="). Compound slots are appended by ".". The actual comment is defined in plain text after the "=". Constraints given in constraint variables will be treated as decision variables.
- Comments for concrete constraints. Here the key is a normalized textual version of the constraint (all variables qualified, "=" escaped by "\=", whitespaces removed). Constraints must be given in the comments file for the model where the comment is applied, e.g., for a refined type the comment shall not be given in the comment file for the project where the refined type is declared rather than in the comment file of the project where the refined type is used in terms of a variable. *Hint:* Try out your model with failing constraints, take the textual version of the respective constraints from the reasoner message and normalize it for the respective comments file.
- Comments for generic constraints. Writing comments for each constraint is tedious, in particular for short constraints such as `isDefined(var);` For such constrains, it's more consistent to define a comment for the constraint in a generic manner, i.e., without operand/arguments as well as comments for the arguments. For the constraint above we can write `isDefined(.) = There is no definition of {0}.` whereby each mandatory argument is given by a dot, potentially separated by commas and the comment may contain placeholders for each argument, starting with `{0}` for the operator, `{1}` for the first argument etc. EASy-Producer translates the arguments first and substitutes the results into the text of the constraint function. However, this may work generically for simple constraints while for complex constraints the 1:1 approach for concrete constraints may be more appropriate for now.

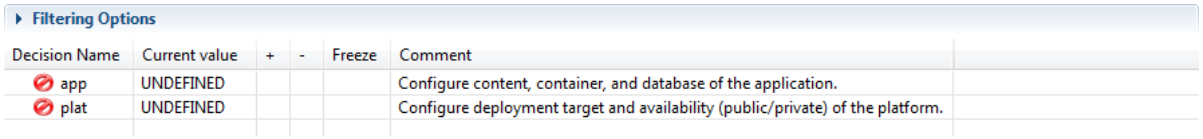
It is also important to mention that there is a comment reuse mechanism for such generic comments. In addition to the project-related comment files as described above, there is a default comment file called `easy-base`, which can be internationalized as the other comment files, e.g., `easy-base.text` and `easy-base_DE.text`.

[9] This is the form for the default comments resource file. A file name may be postfixed by a local language name in order to internationalize the comments, e.g., `PL_Content_Sharing_0_DE.text` for German comments on that model.

The result of the comment-definition is shown in Figure 10. We defined two comments for the two decision variables "app" and "plat" of the running example (cf. Figure 10a)), which are displayed in the **IVML Configuration Editor** of **the Product Line Editor** (cf. Figure 10b)).



a) Comment-definition in the text-file.



b) Decision variables and comments in the IVML configuration editor

Figure 10: Running Example - Definition of Decision Variable Comments.

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.

4.2.2. 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. Their application and combination is defined in a VIL[10] build script. However, some VITs may be realized by an individual instantiator, which actually applies the VIT. A detailed discussion on the concept of instantiators in EASy-Producer can be found in the EASy-Producer developers guide (cf. Section 3.3). 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 11. 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 11, 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 9). Both variables are nested variables of the platform variable `plat`. Thus, they are accessed using "–"-

[10] See the VIL language specification (cf. 3.3) for a detailed description of this language.

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.

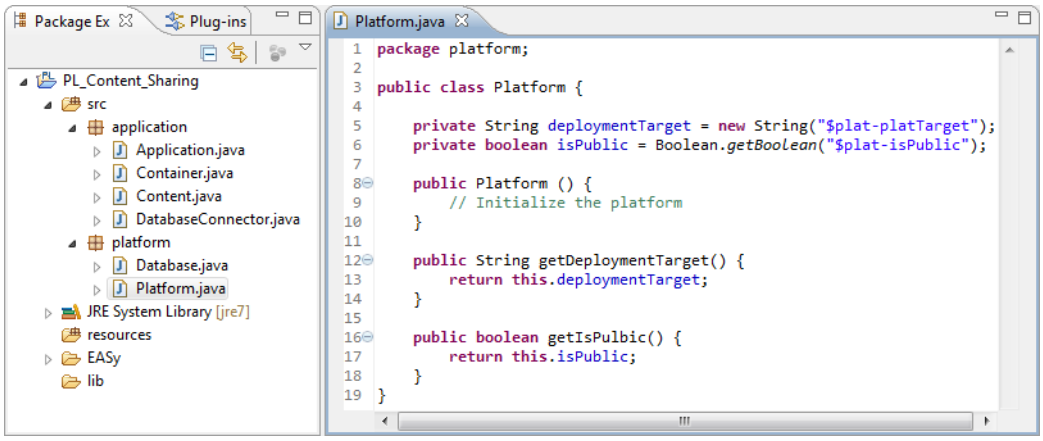


Figure 11: Running Example – The Variability Implementation (snippet).

The next step is to define the VIL build script. Open the **VIL Build Language Editor** by *double clicking* the **VIL-file** in the **EASy-folder**. The file has the name `PL_Content_Sharing_0.vil`. Figure 12 shows the VIL build script of this example. This script is rather simple: the first rule `clean` deletes all source artefacts in the target project (the product project) to guarantee that only the most recent instantiated artefacts are present in the final product. The second rule `main` is the entry-point for the VIL engine. This rule defines `clean` to be a precondition (guaranteeing that the target will be cleaned before the actual instantiation). The only action `main` defines is the call of the Velocity instantiator with the path to the generic source artefacts, the target path, and the configuration as parameters.

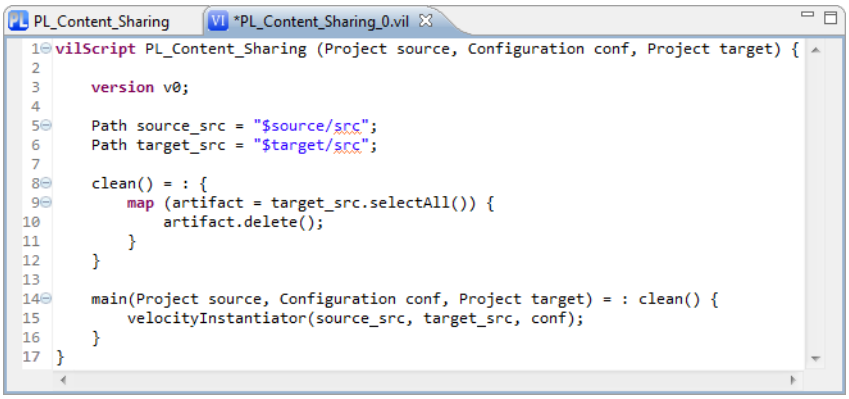


Figure 12: Running Example – The VIL Build Script.

Finally, the implementation space and the corresponding build script are defined 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.

4.3. 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[11] (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 **Project Configuration 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 13.

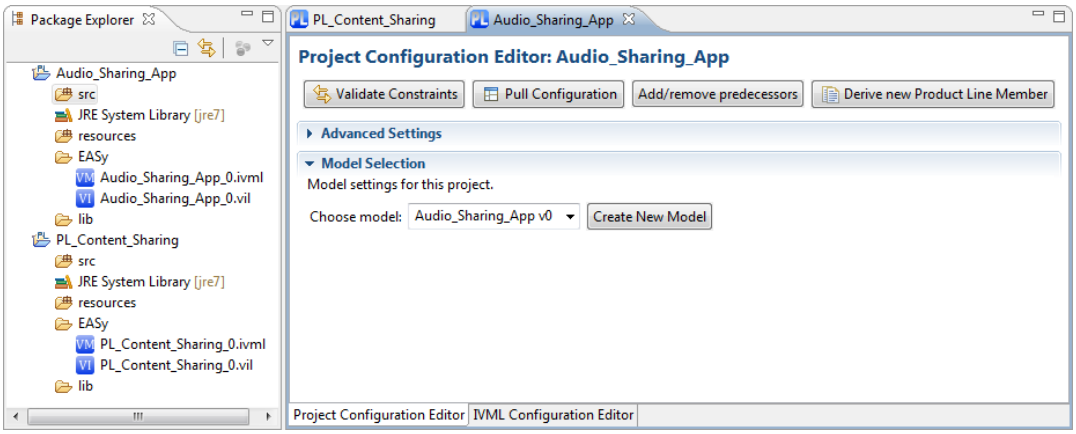


Figure 13: Running Example – The Product Derivation.

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.

4.3.1. 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

[11] 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).

the possible values for each of these elements automatically (we will discuss the configuration editor in detail in Section 5.2.2). Figure 14 illustrates the **IVML Configuration Editor,** including the configurable elements of our audio content-sharing application.

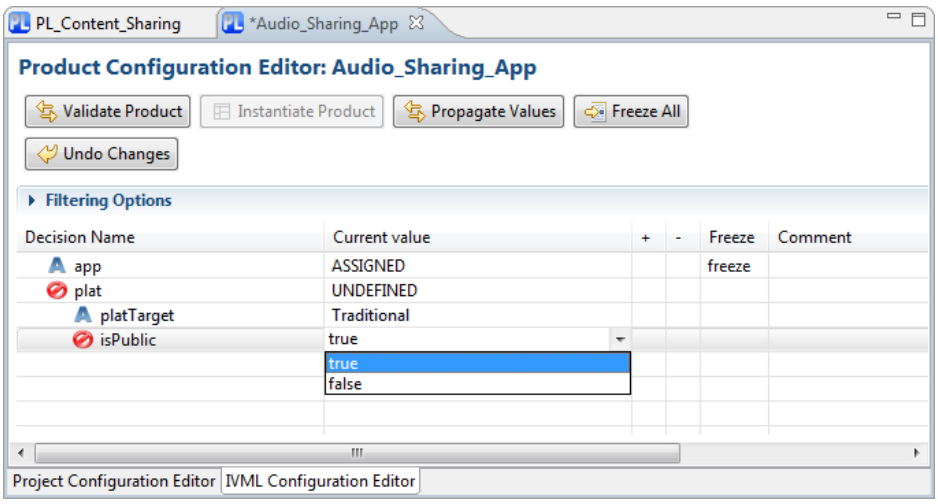


Figure 14: Running Example – The IVML Configuration Editor.

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.

Finally, the product is configured and ready for instantiation.

4.3.2. 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 VIL build script. We defined this information in the previous sections, such as the implementation space and build script (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 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 while resolving the variabilities.

5. EASy-Producer in Detail

In this section, we will describe EASy-Producer in detail. This includes the description of the product line project structure in Section 5.1 as well as the different editors in Section 5.2.

5.1. The Product Line Project Structure

In this section, we will discuss the product line project structure of EASy-Producer. The basic structure of each product line project equals the general structure of Java-project in Eclipse. The only difference is in the EASy-folder of the product line project. This folder contains all EASy-Producer files. These files are:

File Icon	Description
	The IVML-file, which contains the variability model described in the INDENICA Variability Modelling Language, or a specific configuration. This file is **mandatory** and will be automatically created if a new product line project is created.
	The text-file, which contains additional comments for the decision variables defined in the variability model. Please note that we use a "T" for "Text" instead of a "C" for "Comments" as this may be confused with "Configuration". This file is **optional** and has to be created manually.
	The VIL build script file, which contains the specification of the instantiation process of the variable artefacts of the product line project. This file is **mandatory** and will be automatically created if a new product line project is created.
	The VIL template file, which contains the definition of generic templates that can be applied during the instantiation process to create or manipulate specific artefacts and their content. This file is **optional** and has to be created manually.

All files introduced above can be created manually (also those that are mandatory) by clicking *File* → *New* → *Other*. In the wizard, open the **EASy-Producer**-folder and select the file you want to create. Please note that we recommend adding such files to the **EASy-folder** of the product line project as this is the default folder for EASy-specific files. Further, the creation of a variability model, a build script, and templates are supported by individual (text) editors that will open by simply double-clicking the respective file in the **EASy-folder**.

5.2. 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 sub-editors (tabs). This editor opens automatically if a new product line project is created. In order to open the editor manually, *right click* on the product line project and select **Edit Product Line**.

5.2.1. The Project Configuration Editor

The **Project Configuration Editor** provides the general configuration options of a product line project as well as the general actions that can be performed.

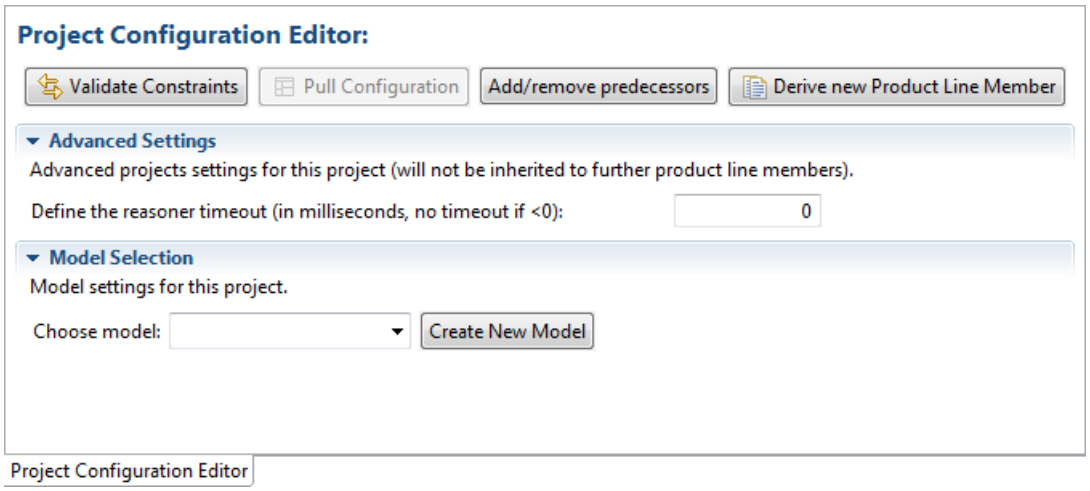


Figure 15: Project Configuration Editor.

Editor Element	Description
"Validate Constraint" Button	Validates all constraints in the selected variability model using the actual reasoner[12].
"Pull Configuration" Button	In case that a new predecessor is added (see below) the configuration of the predecessor is integrated into the configuration of the current product line project.
"Add/remove predecessor" Button	In a Multi Software Product Line scenario (cf. Section 2.3), a new product line project, e.g. a product line product, is initially derived from a single parent product line project, e.g. the base product line. In order to integrate additional parents, this button will open a dialog to select the desired product line projects of the current workspace in Eclipse. Please note that the addition of new parent product line projects also requires the Pull Configuration action (see above) to integrate the configurations of the selected parents into the current product line project.
"Derive new Product Line Member" Button	Derives a new product line project based on the current product line project. For example, this creates a new product project based on the base product line project.

[12] The actual reasoner is determined by the default reasoner setting, which can either be defined by the user or, upon first start, as one of the installed reasoners providing the most reasoning capabilities.

Advanced Settings (Reasoner Timeout)	Restricts the time for the actual reasoner to calculate the validity of a specific configuration to the defined time in milliseconds.
Model Selection	In case that multiple variability models are available, this option enables the selection of the desired variability model as the basis for the configuration of the current product line project.

5.2.2. The IVML Configuration Editor

The IVML Configuration Editor supports the configuration of individual products or partially configured product lines by providing a graphical user interface for the assignment of values to decision variables defined in the variability model. This editor is also used to start the instantiation process after the configuration of a specific product.

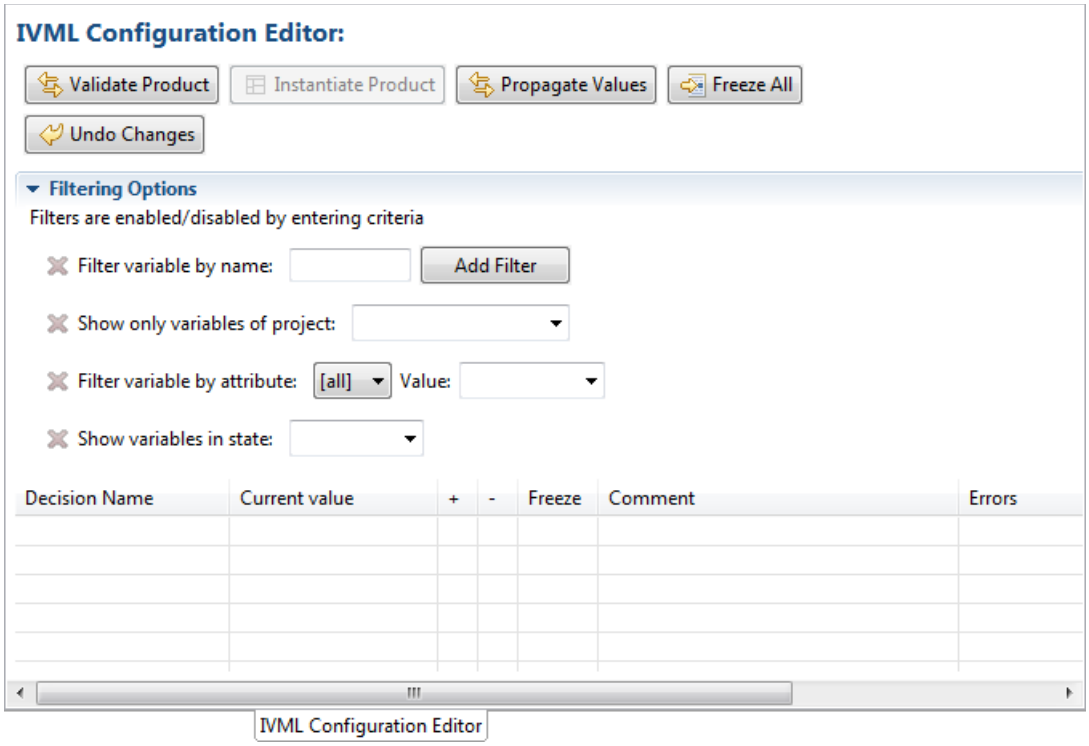


Figure 16: IVML Configuration Editor.

Editor Element	Description
"Validate Product" Button	Validates the current configuration of this product line project using the actual reasoner.
"Instantiated Product" Button	Instantiates the current project line project (partial product line or product) based on the current (partial) configuration.
"Propagate Values" Button	Assigns currently unassigned decision variables of the configuration automatically. This automation

	requires the assignment of a subset of the available decision variables and the relation of these variables to the unassigned variables in terms of constraints in the variability model.
"Freeze All" Button	Freezes all assigned decision variables of the current configuration. The values of frozen decision variables cannot be altered afterwards. For details on the concept of freezing, see the IVML language specification (cf. Section 3.3).
"Undo Changes" Button	Reverts all changes since the last saving of the current configuration.
Filtering Options (Filter by name)	Filters the available decision variables of the current configuration by name.
Filtering Options (Filter by project)	Filters the available decision variables of the current configuration by the project in which they are defined.
Filtering Options (Filter by attribute)	Filters the available decision variables of the current configuration by an attribute and the specific attribute value. For details on the concept of attributes, see the IVML language specification (cf. Section 3.3).
Filtering Options (Filter by state)	Filters the available decision variables of the configuration by their state. The available states are: unassigned, assigned, and frozen.
"Decision Name" Column	The name of the decision variable.
"Current value" Column	The current value of the decision variable.
"+" Column	Adds a new element to a sequence or a set of decision variables.
"-" Column	Deletes an existing element from a sequence or a set of decision variables.
"Freeze" Column	Freezes the decision variable in the row of the current cell.
"Comment" Column	Displays additional information regarding the purpose and the configuration options of the decision variable. This requires the definition of a text-file (cf. Section 4.2.1).
"Errors" Column	Displays errors, for example the violation of a constraint of the variability model by the current value of the decision variable in that row. This

requires the validation of the current configuration.

6. Examples

We provide some examples for running EASy-Producer the first time. It is necessary to install the Examples bundle to execute the examples; the examples will not be available otherwise. The installation of the examples is explained in Section 3.2.

First, we show how to import the examples into the current workspace. In Section 6.2, we briefly explain the examples and how to execute them.

6.1. Importing the Examples into Workspace

In this section, we describe how to import the examples into the current workspace.

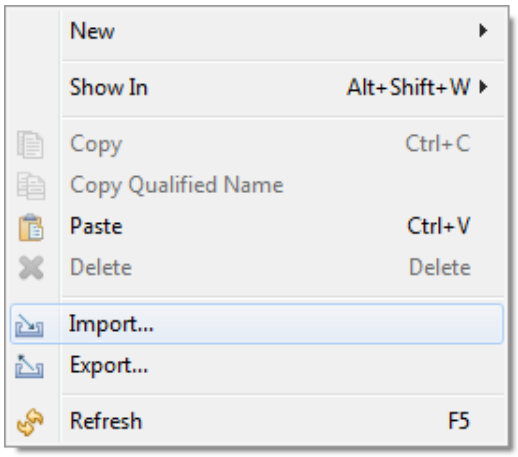


Figure 17: Import selection.

For importing the examples right click into the Package Explorer and select *import*. select first *new* → *Project* (cf. Figure 17). Eclipse asks for the desired wizard. Select here EASy-Producer → Import EASy-Producer Examples to open the Examples Wizard (cf. Figure 18).

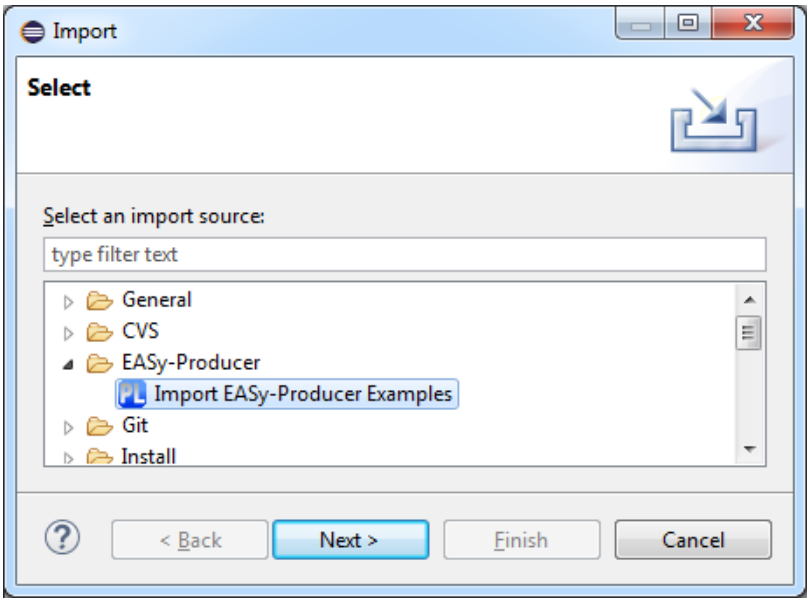


Figure 18: Selection of the Examples Wizard.

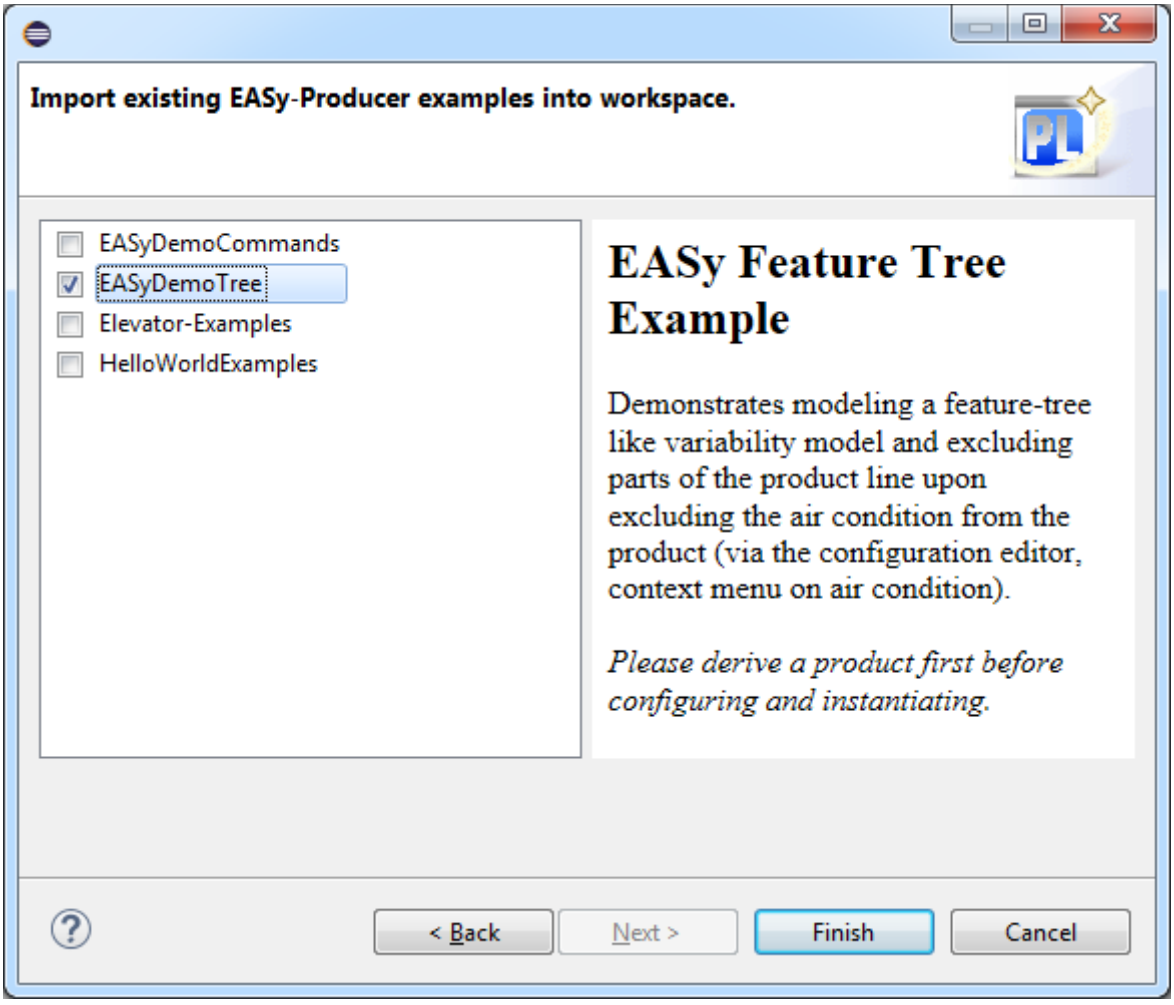


Figure 19: Examples Wizard.

EASy-Producer will open the Examples Wizard to offer possible EASy-Producer examples. Select the desired examples and press on Finish. Each example may install multiple EASy-Producer projects into the current workspace. Importing the examples may take a while.

6.2. Running the Examples

6.2.1. EASyDemoCommands

This example contains several projects, demonstrating different ways how to configure/generate code with VIL.

First, the Product Line editor must be opened to start with one of the contained examples. For doing so, right click on the project inside the package explorer and select "Edit Product Line" (cf. Figure 20).

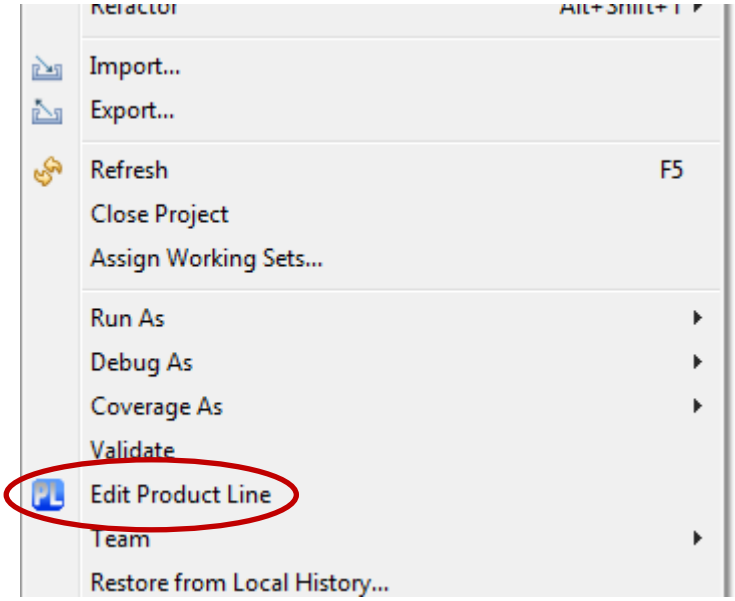


Figure 20: Command to open the "Product Line Editor".

Inside the Product Line Editor on the Project Configuration Editor tab, select "Derive new Product Line Member" to create a new derived projects, which will be used for instantiating a product out of the product line platform. This will open a dialog, which will ask for a name for the new project.

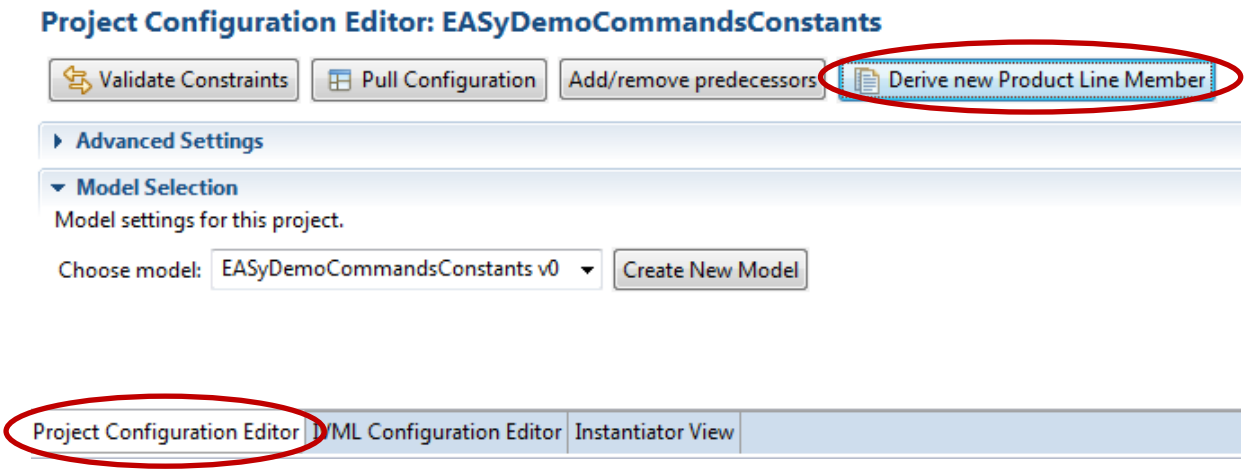


Figure 21: Derivation of a new "product" project.

After entering a new name for the project and pressing the OK button, the project will be created and the Product Line Editor will also be opened for the newly created project. In the following, we briefly describe the different instantiation processes based on the shipped projects.

EASyDemoCommandsPlatform

This project is the parent project of the projects described below. This project is already part of a multi staged configuration process and should not be used for deriving new projects.

EASyDemoCommandsConstants

This project works directly on the Java source files and will transform the existing source code. Figure 22 shows how VIL is used to change the values of defined constants inside the Main class, before the changed classes are compiled into the bin directory.

```
JavaFileArtifact f =
"$target/src/io/ssehub/easy/demo/command/constants/Main.java";
JavaClass cls = f.defaultClass();
cls.attributeByName("APP_NAME").setValue(cfg.appName);
cls.attributeByName("hasAddCommand").setValue(cfg.hasAddCommand);
cls.attributeByName("hasSubCommand").setValue(cfg.hasSubCommand);
cls.attributeByName("hasPrintCommand").setValue(cfg.hasPrintCommand);


javac("$target/src/**/*.java", "$target/bin");
```

Figure 22: VIL code snippet EASyDemoCommandsConstants.

EASyDemoCommandsFile

This project used VIL to delete not selects files from the derived project.

EASyDemoCommandsGen

This project uses VTL to generate new java files according to the specified configuration.

EASyDemoCommandsGenStatic

This project uses VTL to generate one Java source file which constants, which is uses for conditional compilation.

EASyDemoCommandsVelocity

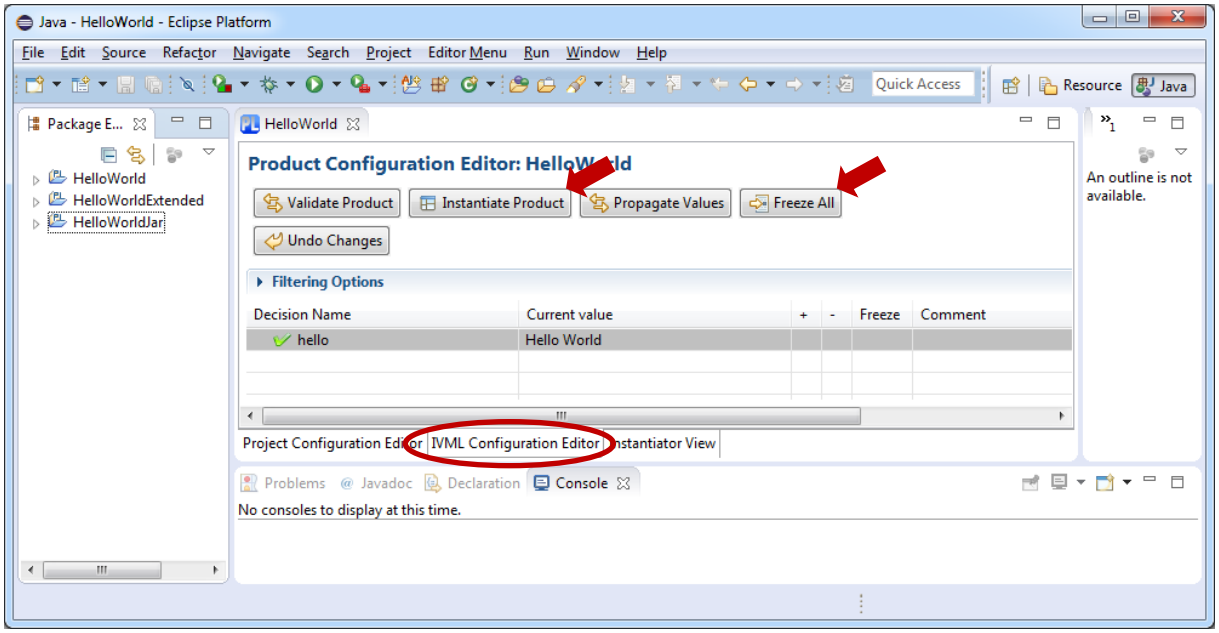
This projects uses velocity[13] as a pre-processor for removing unselected variability out of the source code.

6.2.2. EASyDemoTree

6.2.3. HelloWorld

This example contains three EASy-Producer projects to provide an introduction to VIL. Each project is a self-instantiating project, derivation of new projects is not necessary. We suggest executing the examples in the following order:

1) **HelloWorld** uses Velocity for self-instantiation. This project provides a very simple introduction into VIL, but it can only be instantiated once. Compare the shipped Java source file "HelloWorld.java" before and after instantiation.
2) **HelloWorldExtended** uses VTL to generate code out from a template file. After instantiation, a new Java source file will be generated to "src".
3) **HelloWorldExtended** uses a VTL file to generate code, compile and pack it into a runnable JAR file. After instantiation, a new Java source file will be generated to "src", which will also be compiled to bin and stored inside the newly generated "Hello.jar" file.



For running the examples, right click on the desired project and select "Edit Product Line". Select the "IVML Configuration Editor". Select a value for each variable, click first on "Freeze All" and afterwards on "Instantiate Product".

6.2.4. Elevator-Examples

[13] http://velocity.apache.org/

7. Frequently Asked Questions (FAQ)

In this section, we describe known problems and solutions while working with EASy.

7.1. VIL-Editor won't work after updating EASy/Xtext

Description

After an update of EASy or Xtext, the Editor for VIL scripts displays syntactic errors, even for valid VIL files. In Particular, the editor detects an error inside the first keyword of the file. This is usually the `vilScript` keyword. Also syntax highlighting and the content assist may not work completely. An example is given in Figure 23.

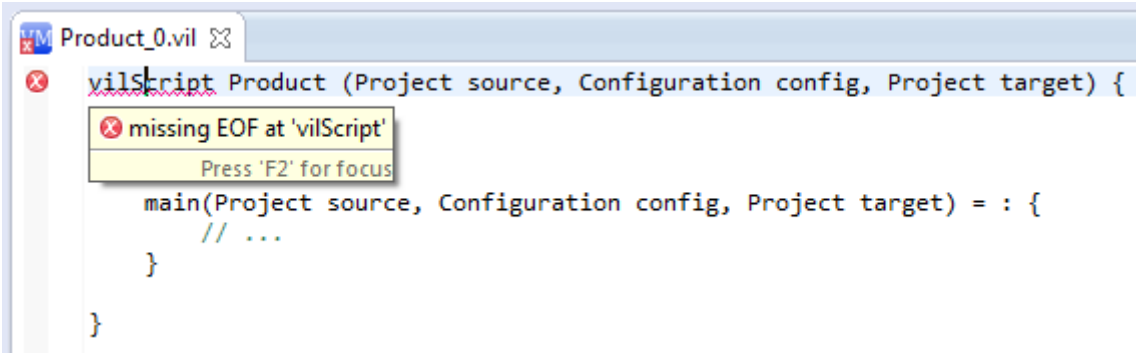


Figure 23: Broken VIL-Editor.

Cause

Eclipse stores information related to its plug-ins inside the workspaces. We observed that this information can cause problems when Xtext was updated.

Solution

Create a new (empty) workspace and import all existing projects into the new workspace using the Eclipse import function. This can be done as follows:

- Open in menu: *File* → *Import…*
- Select: *General* → *Existing Projects into Workspace*
- Select: *Select root directory*
- Click: *Browse*
- Select old workspace and confirm with *OK*
- Select Projects to import (we also suggest to check the option *Copy projects into workspace* (see Figure 24)) and click on *Finish*

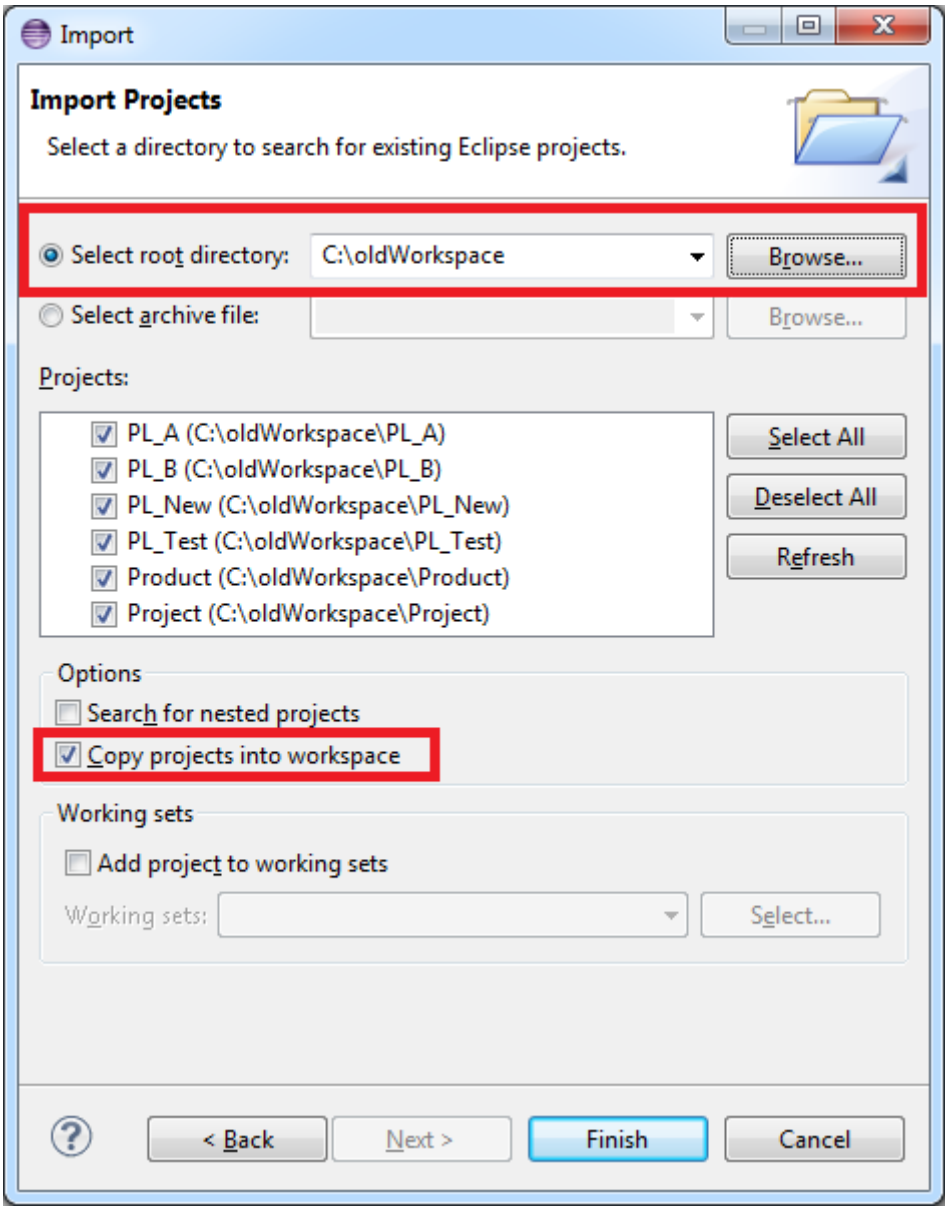


Figure 24: Importing old projects into a fresh workspace

7.2. The Maven integration fails when running it the second time

Description

VIL provides a Maven integration for executing (parts of) the Java build process of a Java product line. Executing the same pom.xml with a packaging stage (Maven assembly plugin) a second time may lead to the message that the system cannot delete the created JAR artifacts.

Cause

Tracing back file open and close calls indicates a problem of unreleased file descriptors in Maven (aether, plexus) and leads to a problem in EASy due to a direct integration of the Maven command line class.

Solution

Prior to EASy-Producer 0.11.4 / Maven integration 3.2.3.1 the only solution is to close EASy after executing the instantiation. Newer versions solve this problem by integrating Maven as a standalone process. However, if the standalone process cannot be executed due to an unexpected environment (the integration has to reconstruct the actual classpath), you may switch back to the direct integration setting the JVM system property `easy.maven.asProcess` to `false` (i.e., `-Deasy.maven.asProcess=false`), which potentially leads to the problem described above.

7.3. The Maven integration does not execute at all

Description

The VIL-Maven integration shall be installed in EASy-Producer as an unpacked bundle, i.e., not as a single Jar file.

Cause

The capability of unpacking a single bundle due to its Manifest specification depends on the underlying Eclipse version (as of version 3.3, April 2011). However, it may be that Eclipse does not recognize the unpacking directive.

Solution

Either you unpack the VIL-Maven integration bundle manually (close Eclipse, go to the plugins folder, unpack the most recent bundle Jar with name starting with de.uni_hildesheim.sse.easy.instantiator.maven, delete the Jar and start Eclipse again) or you switch back to the direct integration, potentially preventing multiple sequential executions of Maven as described in Section 7.2.